Network Accounting

Ein benutzerbezogenes Abrechnungssystem für die Nutzung von TCP/IP-Diensten in Linux-Systemen

Markus Heidenreich

Kaiserslautern,
Juli 2000

Projektarbeit
Fachbereich Informatik
Universität Kaiserslautern

Betreuer :
Prof. Dr. R. Gotzhein (AG Rechnernetze)
Dr. W. Eicher (RHRK)

Inhaltsverzeichnis

1. Einleitung

Gerade in einer Zeit, in der das Internet in nahezu alle Bereiche des menschlichen Lebens vorgedrungen ist und sich nicht zuletzt aufgrund seiner unbegrenzt scheinenden Möglichkeiten zur Beschaffung und zum Austausch von Informationen und zur weltweiten Kommunikation eines sehr starken Zuspruchs erfreut, liegt es nicht nur im Sinne von Rechenzentren und Dienstanbietern, eine Möglichkeit zur Abrechnung der in Anspruch genommenen Ressourcen in die Hand zu bekommen.

Die Erschließung neuer Regionen, sowie der Ausbau vorhandener Netze in Richtung einer Bereitstellung höherer Bandbreiten zur Verbesserung der Übertragungsgeschwindigkeiten ist mit immensen Kosten verbunden.

Es ist nicht Aufgabe dieser Arbeit zu entscheiden, auf welche Art und Weise die Kosten auf die Benutzer umgelegt oder verteilt werden sollen. Wir wollen hier auch keine Vorschläge zu solchen Überlegungen einbringen, da dergleichen die Domäne anderer Disziplinen, wie beispielsweise der Betriebs- und Volkswirtschaftslehre und der Politik, darstellt.

Unsere Aufgabe ist es aber, die informatikspezifischen Probleme der rechnerinternen Erfassung von Accountinginformationen zu beleuchten und so gesammelte Werte den Spezialisten anderer Fachgebiete zur weiteren Verarbeitung zu überlassen.

So befasst sich diese Arbeit zunächst mit den grundlegenden Eigenschaften und Modellen des zu betrachtenden Datenverkehrs, um im folgenden Voraussetzungen und Möglichkeiten zur Realisierung einer benutzerorientierten Erfassung und Abrechung der genutzten Netzwerkressourcen aufzuzeigen und herauszuarbeiten.

2. Aufgabenstellung

Die Aufgabe dieser Arbeit stellt sich im einzelnen wie folgt dar :

Grundsätzliches Ziel ist die Konzeption und Realisierung eines Systems zur benutzerbezogenen Erfassung und Abrechnung von Netzwerkaktivitäten in Linux-Systemen.

Erreicht wird das Ziel durch Implementierung und Untersuchung einer kernel-erweiternden Software, die auf einem Linux-Rechner die Netzaktivitäten einzelner Benutzer protokolliert, so dass ein benutzerspezifisches Network-Accounting auf dem Rechner erfolgen kann.

Die hierbei gesammelten Informationen sollen in Protokoll-Dateien aufgenommen und in der notwendigen anonymisierten Form, d.h. nur in Abhängigkeit der Quelladressen bei Unterdrückung der Zieladressen, dargestellt werden. Hier dient das systeminterne Wissen über die Zieladressen lediglich einer selbst definierbaren Gewichtungsvergabe für die jeweils ausgetauschten Daten, aber nicht einer expliziten Ausgabe und Bekanntmachung.

Prinzipiell könnte es als sinnvoll erscheinen, eine Accounting-Software auf einem Router zu implementieren, da man hier die maximale Datenausbeute hat, d.h. den gesamten ein- und ausgehenden Verkehr des dahinterliegenden Netzes erreichen und auswerten könnte. Es ergibt sich an dieser Stelle aber das unüberwindbare Problem, dass hier kein benutzerbasiertes Accounting in Bezug auf Mehrbenutzersysteme durchgeführt werden kann, da der Router zwar sowohl Sender- als auch Empfängeradresse (Rechneradresse auf IP-Ebene) der zu übermittelnden Daten "sieht", aber keine Kenntnis über die lokalen Benutzernamen hat.

Die Zuordnung von Transfereinheiten zu lokalen Benutzernamen reicht nicht über die Grenzen des Kernels des jeweilig verwendeten Betriebssystems heraus. Damit verliert sich die Möglichkeit der Ermittlung des Verursachers, sobald eine Transfereinheit den Rechner verlassen hat. Aus diesem Grund ist es nicht möglich, mit Hilfe eines sogenannten "Leitungs-Sniffers" die Aufgabe des benutzerbezogenen Network-Accounting von einem beliebigen Punkt im Netzwerk zu bewältigen, man muss sich stattdessen auf jeden einzelnen Rechner im Netz zurückziehen, um hier jeweils aus den Betriebssystemkernen heraus zu agieren.

Für die verbindungsorientierte Art des Datenaustauschs wäre es sicherlich hilfreich zu erkennen und auszuwerten, wer die Verbindung aufgebaut hat, um demjenigen als "Schuldigen" die durch den Transfer erzeugte Last zuzuordnen.

Zunächst soll eine Marktrecherche bezüglich freier sogenannter Public-Domain-Software durchgeführt werden, um das Ziel eines benutzerbezogenen Accounting zu erreichen. Schließlich ist es nicht notwendig, jede Idee auf die gleiche Art und Weise mehrmals neu zu entwickeln und zu realisieren.

3. Grundlagen des IP-Verkehrs

Zur Beschreibung der Datenübertragung zwischen entfernten Rechnern über das Internet setzen wir an dieser Stelle ein hybrides Modell der TCP/IP-Suite und dem OSI-Referenzmodell (Abbildung 3.1) als bekannt voraus. Im Hinblick auf unsere Aufgabe eines Network-Accounting unterscheiden wir bezüglich Ebene 4 zwischen *verbindungsorientierter* und *verbindungsloser* Art der Übertragung.

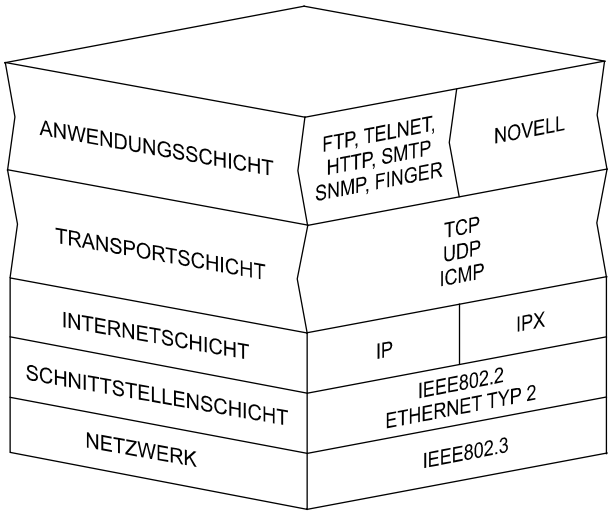

Abbildung 3.1: Hybridmodell (in Anlehnung an [4])

Bei der verbindungsorientierten Variante wird zur Lenkung der auszutauschenden Nachrichten, also zu deren Transport, das *Transmission Control Protocol* (*TCP*) benutzt. Hier läuft der Datentransport in den folgenden drei Phasen ab :

- *Verbindungsaufbau :*
  der als Dienstnutzer (*Client*) fungierende Rechner baut eine Verbindung mit dem Dienstanbieter (*Server*) auf, wobei sich die Kommunikationspartner über Parameter einigen können.

- *Datenübertragung :*
  unter Benutzung der aufgebauten Verbindung werden Nachrichten ausgetauscht, wobei hier die explizite Angabe von Absender und Empfänger nicht notwendig ist, da die Eintragung der Verbindungsnummer von Endteilnehmer zu Endteilnehmer die Partner über das Netz hinweg eindeutig bestimmt.

- *Verbindungsabbau :*
  die Verbindung zwischen den Partnern wird gelöst und evtl. dafür reservierte Ressourcen werden freigegeben.

Für die verbindungslose Übertragung steht das Transportprotokoll *User Datagram Protocol* (*UDP*) zur Verfügung. Auch das für Meldungen über Netzzustände verwendete *Internet Control Message Protocol* (*ICMP*) arbeitet verbindungslos.

Wie der Name vermuten lässt, findet hier vor der Datenübertragung kein Verbindungsaufbau statt. Die Nachrichten müssen folglich hier vollständige Angaben über Absenderadresse und Zieladresse enthalten.

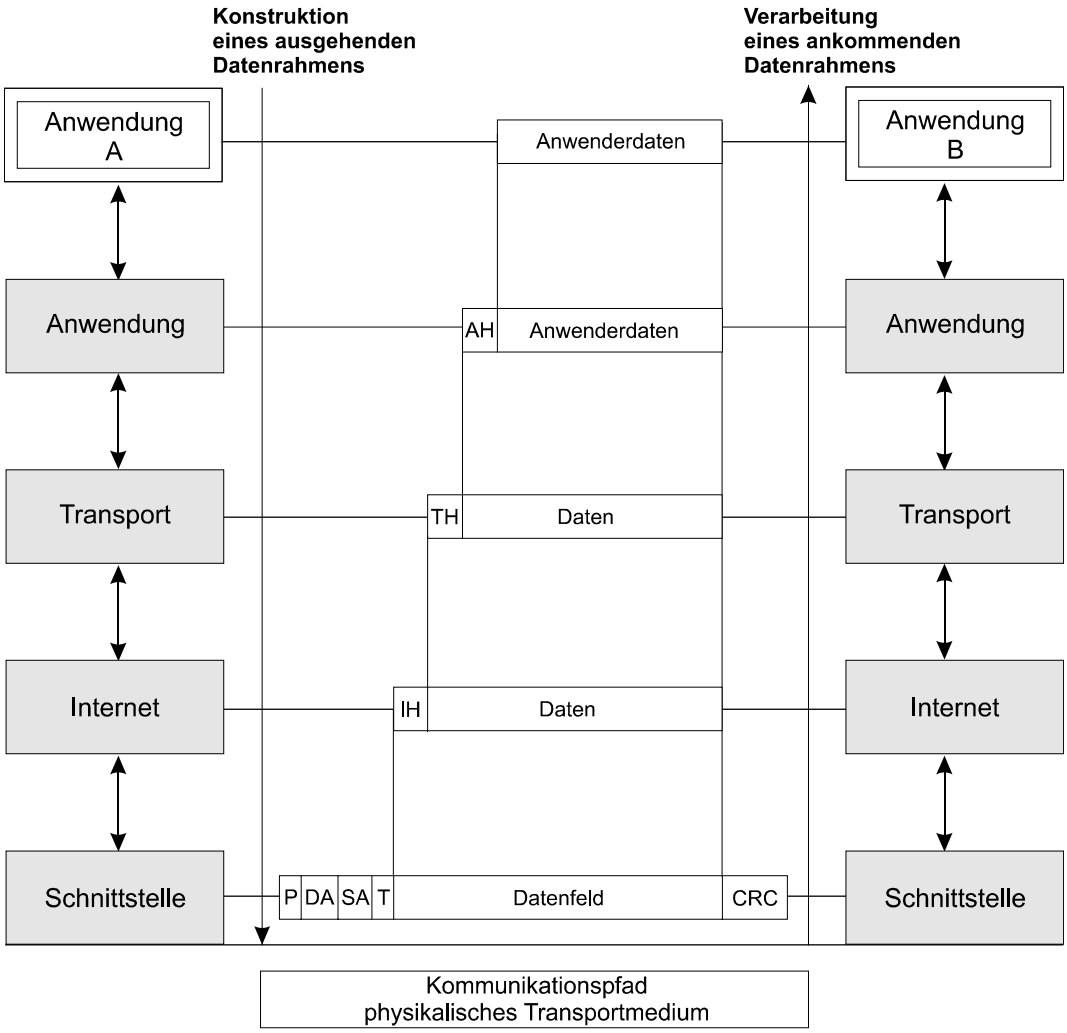

Abbildung 3.2: Bau, Versand und Auswertung eines Datenrahmens nach dem Hybridmodell (in Anlehnung an [4])

Da die Kommunikation über das Internet ursprünglich auf dem verbindungslosen Ansatz basiert, enthält jedes übertragene Paket selbst bei verbindungsorientiertem Transport die vollständigen Informationen über Absender- und Zieladressen auf der Internetebene (IP-Ebene), was sich für die Realisierung des Accounting-Systems als sehr hilfreich erweisen wird.

Wie aus der obigen Abbildung 3.2 zu erkennen ist, durchläuft eine Nachricht zunächst auf der Absenderseite alle Schichten von oben nach unten, wird schließlich über das Netz transportiert, um auf der Seite des Empfängers die Schichten auf dem Weg zur Anwendung nach oben zu durchlaufen.

Zwischen jeder Schicht wird das Prinzip der *Encapsulation* realisiert, d.h. jede Schicht verpackt eine Transfereinheit der jeweils höheren Schicht, setzt einen eigenen *Kopf* (*Header*) mit eigenen Informationen vor deren Anfang und reicht sie an die tiefere Schicht zur Bearbeitung weiter. Empfängerseits setzt der umgekehrte Prozess ein und bis zum schließlichen Auspacken der letztendlichen Nutzdaten durch die Anwendungsschicht entfernt jede Schicht den zugehörigen Header, wertet diesen aus und reicht den Rest der Transfereinheit an die nächsthöhere Schicht weiter. So liegt die komplette Einheit einer übergeordneten Schicht im Datenteil der jeweils betrachteten Schicht vor.

Zur benutzerorientierten Erfassung des hier transportierten Datenverkehrs scheint es offenbar zu genügen, sich an einer geeigneten Stelle in den Schichtenstapel "einzuklinken" und dort aus jedem zu behandelnden Paket die interessierenden Daten auszulesen und in selbst zu führende Protokolldateien aufzunehmen.

Würde man sich nun von der Netzwerkseite her in diesen Schichtenstapel so einklinken, dass man im Netz jede, beispielsweise einen Router, passierende Transfereinheit abfängt und einer Art "Verkehrskontrolle" unterzieht, so kann man - wie untenstehende Veranschaulichung der jeweiligen Header verschiedener Schichten (Abbildung 3.3) zeigt - zwar recht problemlos z.B. Herkunfts- und Zieladresse, sowie Größe des herausgewunkenen Fahrzeugs auslesen, erhält allerdings hier keinerlei Informationen über einzelne Benutzer, die sich gerade in Mehrbenutzersystemen zahlreich hinter den gelesenen Netzwerkadressen verbergen. Wie bereits in Abschnitt 2 bemerkt, sollte man für die aktuellen Betrachtungen nicht an "personenbezogen" vergebene IP-Adressen denken, denn in Umgebungen, in denen es möglich ist pro Benutzer eine IP-Adresse zu vergeben ist das Accounting ohnehin wesentlich erleichtert und bedarf nicht dem zu betreibenden Aufwand, wenn sich viele verschiedene Benutzer gleichzeitig anmelden und weitere Netzwerkaktivitäten von dieser Stelle aus tätigen.

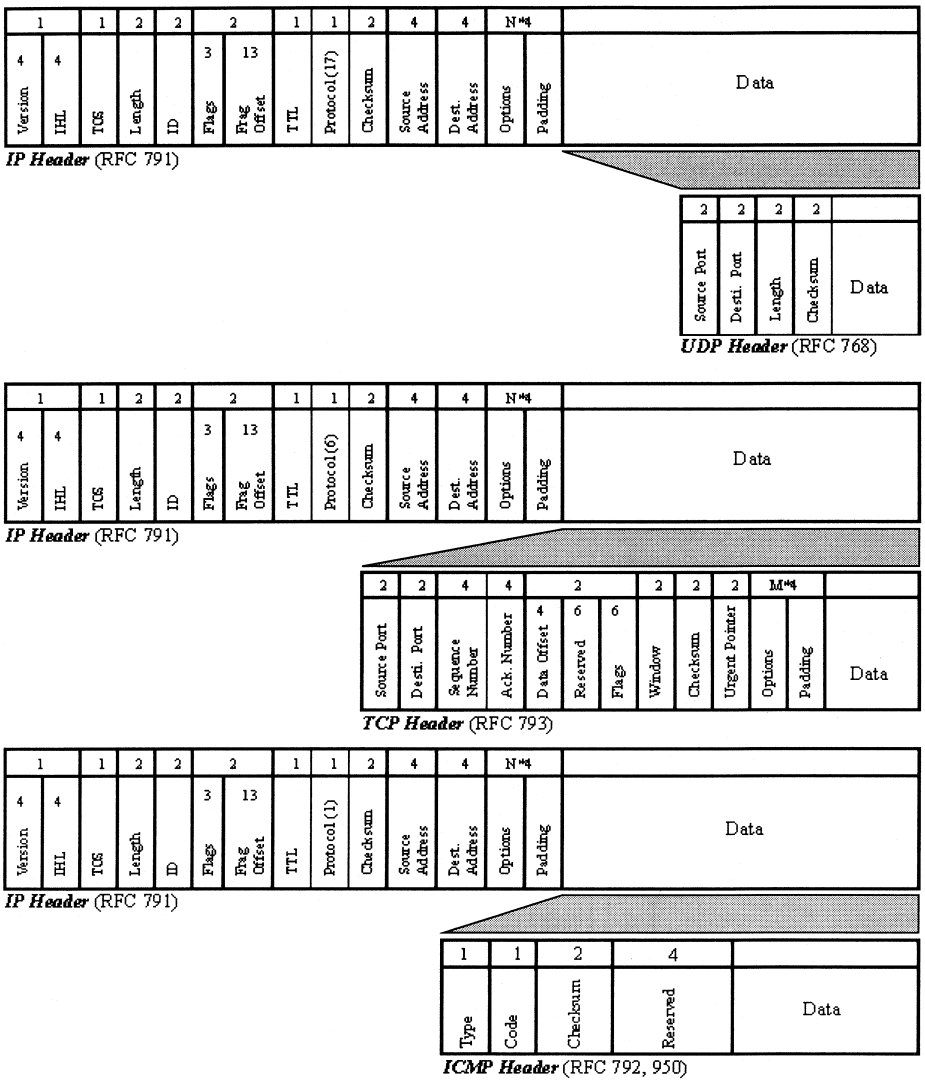

Abbildung 3.3: Header der Protokolle auf Transport- und Internetebene [1]

Es ist also notwendig, den Eingriff in den Schichtenstapel nicht von außerhalb - aus der Position des Netzwerks - vorzunehmen, sondern direkt im Betriebssystem beim Abarbeiten der Schichten einzuhaken, da hier, wie in Abschnitt 2 beschrieben, die einzige Möglichkeit besteht, zu transportierende oder ankommende Dateneinheiten den lokalen Benutzern zuzuordnen.

Es soll hier nicht eine philosophische Diskussion angeregt werden, wie das Betriebssystem und das Hybridmodell zusammenarbeiten, ob das Betriebssystem als Teil des Referenzmodells angesehen wird oder umgekehrt oder ob es neben dem Modell steht. Wir wollen diejenigen Betriebssystemteile, die für die Netzkommunikation zuständig sind, lediglich als Instanzen des im Modell vorgegebenen Konzeptes betrachten und folglich das Modell lediglich als beschreibendes Mittel.

Insofern befindet sich das Betriebssystem in jeder Schicht, sowie an den Übergängen zwischen den Schichten, da es die durch das Modell gegebene Beschreibung weitgehend detailgetreu realisiert.

4. Die Accounting-Umgebung

Die Kenntnis und Dokumentation der Accounting-Umgebung, also des verwendeten Betriebssystems, ist außer der letztendlichen Realisierung des Abrechungssystems eine wichtige Aufgabe, denn je nach Beschaffenheit des Betriebssystems und der Kenntnis vom Zusammenhang der einzelnen Komponenten entscheidet sich, an welchen Stellen Ausgaben aus dem Kern des Betriebssystems zu tätigen sind, um eintreffenden und ausgehenden Netzverkehr den einzelnen Benutzern zuordnen zu können.

Man muss Zugriff auf die Quelltexte des Betriebssystems besitzen, um zunächst lesend entscheiden zu können, an welchen Stellen später schreibend Eingriffe vorzunehmen sind.

Nicht zuletzt wegen dieser Voraussetzung des Offenliegens des gesamten Quellcodes, werden wir im Rahmen dieser Arbeit im folgenden das Betriebssystem Linux untersuchen. Ohne rechtliche Einschränkungen dürfen wir hier die unter der "GNU Public License" entwickelten Linux-Quellen frei modifizieren und die verwendete Hardwareausrüstung ist durch die Verwendung eines Standard-PCs möglichst anschaffungsgünstig. Darüber hinaus können wir uns sicher sein, dass wir bei der momentan wohl täglich wachsenden Anhängerzahl mit Linux auf keinen Exoten unter den PC-Betriebssystemen setzen.

Wir wollen hier eine der gängigsten Linux-Distributionen aus dem Hause SuSE betrachten. Der in SuSE Linux 6.3 mitgelieferte Kernel ist der von Linus Torvalds (dem Linux-Pionier schlechthin) herausgegebene Standard-Kernel in Version 2.2.13, der in seinem vollständigen Quelltext ausgeliefert wird. Durch eigene Veränderungen an diesen Quelltexten, die in der Programmiersprache C abgefasst sind und anschließender Kompilierung erreicht man zusätzlich gewünschtes Systemverhalten.

Zur wesentlichen Erleichterung einer solchen Modifikation verfügt Linux über mächtige Werkzeuge, wie einer menügeführten Konfiguration und Zusammenstellung der zuvor implementierten Kernelkomponenten und dem sehr variabel einsetzbaren Werkzeug "make" zur skriptbasierten Kompilierung der Komponenten und Module, das aus der UNIX-Welt stammt und dort altbekannt ist.

Linux als Multiuser-System ermöglicht es, dass sich mehrere Benutzer gleichzeitig anmelden und parallel Ressourcen nutzen können. In einer solchen Mehrbenutzerumgebung muss das Accounting betriebssystemseitig gesteuert werden, wogegen es in einer Einbenutzerumgebung, wie beispielsweise einer Windows NT Workstation, an der sich gleichzeitig nur ein Benutzer anmelden kann, genügen würde, ein systemweites Netzlastprotokoll mit dem Loginprotokoll mit Hilfe eindeutiger Datums- und Uhrzeitdarstellungen in Verbindung zu bringen und so die erzeugte Last dem jeweils angemeldeten Benutzer zuzuordnen.

5. Durchführung des Accounting

Wie schon in 4. beschrieben, spielt sich die Steuerung der Netzwerkaktivitäten (einschließlich der Transportschicht abwärts) zentralisiert im Betriebssystemkern ab. Deshalb wird es möglich, jede Übertragung unabhängig von der erzeugenden Anwendung zu protokollieren.

Für das Senden und Empfangen von Netzwerkpaketen ist das Öffnen eines sogenannten "Sockets" von Nöten, das man sich wie einen Portier am Ein- und Ausgang vorstellen kann. Ein solches Socket ist immer in der Eigentümerschaft eines Benutzers und bis auf wenige Ausnahmen, die später noch dargestellt werden, besitzt es derjenige Benutzer, der letztendlich Daten sendet bzw. empfängt.

Diese Eigentümerschaft von Sockets wird auch im Betriebssystemkern verwaltet und ermöglicht die Zuordnung eines jeden Datenpakets, dessen Länge man sogar im Paketkopf (Header) auslesen kann, gerade zu dem Socketbesitzer. Führt man nun eine Tabelle, in der die einzelnen Benutzer durch ihre *Benutzernummer* (*User ID*) eindeutig repräsentiert sind und addiert pro Benutzer die durch ihn hervorgerufenen Paketlängen, so ist der Schritt zur Erfassung der Accounting-Daten getan.

Man kann sich dann überlegen, wie man diese gesammelten Daten in geeigneter Form ausliest und textuell oder graphisch zur Anschauung aufbereitet.

Der in Anhang 8.1. aufgezeichnete Quelltext des Patches für den Linux-Kernel 2.2.13 nimmt an den entsprechenden Stellen im Betriebssystemkern Ergänzungen vor, die die oben genannten zusätzlichen Aufgaben des Sammelns und Erfassens der Accounting-Daten leisten. Im einzelnen werden folgende Passagen hinzugefügt :

- Documentation/Configure.help (wird erweitert) :
  Es handelt sich hier lediglich um eine verbale Kurzbeschreibung der Kernelkomponenten mit dem Zweck der Information desjenigen Benutzers, der für Kernelkonfiguration zuständig ist, über die Funktionsweise und Wirkung.

- arch/i386/defconfig (wird erweitert) :
  Hier wird eine Vorauswahl für die Konfiguration und Zusammenstellung der Kernelkomponenten vor der Kompilierung definiert. Standardmäßig ist das Einbinden des benutzerbezogenen IP-Accounting ausgewählt.

- include/linux/ip_acct_user.h (kommt neu hinzu) :

  Hier werden alle später benötigten Datenstrukturen und Konstanten vereinbart. Beispielsweise besteht ein Eintrag in der Gewichtungstabelle (struct ip_acct_weight_s) aus einer Adresse (addr), einer Maske (mask), dem zugehörigen Gewicht (weight) und dem Zeiger auf den nächsten Tabelleneintrag.

- include/linux/proc_fs.h (wird erweitert) :

  Hier werden grundsätzlich Eigenschaften und Erscheinungsbild des /proc-Dateisystems festgelegt. Es stellt ein virtuelles Dateisystem dar und beinhaltet textuell repräsentierte Extremitäten des Kernels. Durch den Eintrag zweier weiterer virtueller Dateien kann dieser Vorzug des /proc-Dateisystems genutzt werden, da man ohnehin im Kernel zu agieren hat und so den recht unkomfortablen Umgang mit realen Dateien vermeiden kann.

- net/ipv4/Config.in (wird erweitert) :

  In dieser Datei finden sich die Definitionen vieler bool'scher Variablen, die bei einer Konfiguration der Kernelkomponenten entsprechend der Auswahl auf "wahr" oder "falsch" gesetzt werden. Natürlich wird hier für das "IP: Per user accounting" ebenfalls eine Variable eingeführt.

- net/ipv4/Makefile (wird erweitert) :

  Das Makefile, das vom bereits beschriebenen Kompilierungstool "make" benutzt wird, entscheidet je nach Belegung der in der oben genannten "net/ipv4/Config.in" definierten Variablen, welche Kernelkomponenten bei letztendlichem Erstellen des Kernels einbezogen werden. Eingefügt wird hier die Abfrage unserer "CONFIG_IP_ACCT_USER"-Variable mit der notwendigen Konsequenz.

- net/ipv4/af_inet.c (wird erweitert) :

  An dieser Stelle des Kernels ist vorwiegend das E/A-Verhalten von Sockets definiert, teilweise sind hier auch Funktionen und Routinen enthalten (z.B. void inet_proto_init(...)), die beim Systemstart aufgerufen werden und die Implementationen der Protokolle starten. Hier lassen wir ebenso eine Routine void ip_acct_user_init(void) ablaufen, die in der ip_acct_user.c definiert ist und wichtige Voraussetzungen, wie die Spezifikation der beiden Extremitäten im /proc-Dateisystem und die Initialisierung der Gewichtungstabelle bereitstellt. Desweiteren werden mehrere Socketkommandos hinzugefügt, die sich selbst bei ihrem Aufruf einer Funktion int ip_acct_user_ioctl(...) als Argument übergeben. Diese Funktion findet sich auch in der ip_acct_user.c.

- net/ipv4/ip_acct_user.c (kommt neu hinzu) :

  Hier befindet sich ein Herzstück des benutzerbasierten Netzwerk-Accounting. Es werden alle notwendigen Routinen bereitgestellt, die eine komplette Verwaltung der Gewichtungstabelle, repräsentiert durch /proc/net/ip_acct_weight, und der Benutzertabelle, die in /proc/net/ip_acct_user bereitgehalten wird, ermöglichen, sowie die zentralen Funktionen, die das letztendliche Erfassen der Accountingdaten durchführen. Die bereits angesprochene Initialisierung und Befehlsverarbeitung zu Systemstart- und Laufzeit ist im wesentlichen ebenfalls hier implementiert.

- net/ipv4/ip_output.c (wird erweitert) :

  Für ausgehenden Netzverkehr wird hier für alle Transportprotokolle (TCP, UDP, ICMP) zentral die Erfassung der Accountingdaten eingefügt. Als kleines Zusatzgeschenk ist es hier möglich, für einzelne Benutzer die Berechtigung zu Netzaktivitäten zu unterbinden. Dies ist steuerbar durch einen manuell einfügbaren Eintrag in der Benutzertabelle, der hier abgefragt wird und ggf. die Senderoutine frühzeitig beendet. Ist der Benutzer nun berechtigt, so wird später die Funktion int ip_acct_user_sent(...), die sich in der ip_acct_user.c befindet, mit den Parametern UserID (sofern diese ermittelbar ist, falls nicht wird der Benutzer "Nobody" herangezogen), Zieladresse und Paketlänge mit dem Schreiben dieser Accountingdaten beauftragt.

- net/ipv4/raw.c (wird erweitert) :

  Die drei letzten betrachteten Quellen stellen im wesentlichen Implementierungen von Transportprotokollen dar. Hier wird analog zur ip_output.c die aus der ip_acct_user.c stammende Funktion int ip_acct_user_received(...) zwischengeschaltet und während der Empfangsroutine mit dem Schreiben der Accountingdaten beauftragt.

- net/ipv4/tcp_ipv4.c (wird erweitert) :

  siehe raw.c

- net/ipv4/udp.c (wird erweitert) :

  siehe raw.c

Zusammenfassend werden die wesentlichsten Ergänzungen in denjenigen Routinen vorgenommen, die für die Erstverarbeitung empfangener bzw. für das Abgeben zu sendender Transfereinheiten verantwortlich sind. Weitere Änderungen und Ergänzungen sind eher verwaltungstechnischen Charakters, um die Benutzer- und Gewichtungstabelle im /proc-Dateisystem, sowie eine bequeme Kompilierung der gepatchten Kernelquellen zu ermöglichen.

Der Kernel-Patch sowie ergänzende externe Steuerprogramme sind ursprünglich unter der Leitung von Ramses Smeyers (smeyers@khk.org) entstanden. Die sogenannte "UserIPacct Page" befindet sich unter http://www.sin.khk.be/ipacct/. Hier findet sich die aktuellste Version mit einigen wenigen Beschreibungen zum Download. Die in Abschnitt 2 angesprochene und hier durchgeführte Marktrecherche soll kurz beschrieben werden. Eine für Linux-Ressourcen allgemein sehr gut geeignete und mächtige Suchmaschine ist verfügbar unter http://www.freshmeat.net, die, nach langwieriger Recherche in allen bekannten Allgemeinsuchmaschinen, letztendlich aufgrund mehrerer Gespräche im *Internet-Relay-Chat* (*IRC*) und dort innerhalb des Channels "linux.de" und der Nutzung einiger Linux-spezifischer Newsgroups ausfindig gemacht wurde.

Nachdem der Kernel-Patch die erfassten und gesammelten Accounting-Daten dem Benutzer im */proc-Dateisystem*, das ein virtuelles Dateisystem darstellt und die Extremitäten des Kernels in Textform wiedergibt, in recht unlesbarer Art und Weise präsentiert, übernimmt ein externes ausführbares Programm die Interaktion zwischen Benutzer und den ermittelten Accounting-Daten. Über dieses Programm ist es möglich, eine Gewichtstabelle, die auch über das */proc-Dateisystem* bereitgestellt wird, virtuell zu editieren und somit für verschiedene Kommunikationspartner bzw. gesamte Subnetze verschiedene Gewichtungen zu definieren.

Beispiele für die beiden zusätzlichen Kernelextremitäten möchten wir im folgenden kurz betrachten und erklären.

Die Benutzertabelle beinhaltet, wie das folgende Beispiel zeigt, die Benutzernummer, die gesendeten und empfangenen Byte und ein Flag, das am Ende dieses Abschnitts noch erklärt wird.

```
uid     sent      recv      flags
0       173064    804972    00
30      148490    8269      00
65534   3424      0         00
500     83049     543782    00
```

Die Gewichtungstabelle zeigt sowohl Adresse, als auch Maske in Hexadezimaldarstellung, wobei die Zahlenpaare in umgekehrter Reihenfolge und Dezimaldarstellung aufgelöst eine IP-Adresse in gewohnter Schreibweise repräsentieren. Zum Beispiel löst sich die Adresse `36A6F683` auf zu `131.246.166.54` und die Maske `0000FFFF` steht für `255.255.0.0`. So bedeutet die zweite Zeile der Gewichtstabelle, dass der Verkehr im Subnetz `131.246.*.*` mit dem Gewicht `0` bewertet wird, also ist der Verkehr in diesem Subnetz "gebührenfrei".

```
addr            mask            weight
0100007F        FFFFFFFF        0
36A6F683        0000FFFF        0
00000000        FFFFFFFF        1
```

In Anhang 8.2 findet sich der Quellcode für ein solches externes Steuermodul.

Wie oben bereits gesehen werden in der /proc/net/ip_acct_user die einzelnen Benutzer über ihre numerische User-ID repräsentiert, was direkt in dieser Darstellung nicht komfortabel zu lesen ist. Deshalb setzt die Ausgabe des externen Steuerprogramms diese numerische Repräsentation in den jeweiligen alphanumerischen Benutzernamen um.

In einem ähnlich unkomfortablen Format werden die IP-Adressen und Subnetz-Masken in der Tabelle /proc/net/ip_acct_weight angezeigt, die auch durch das externe Programm in die gewohnte Form zur Ausgabe übersetzt werden.

Zum Überblick über die genaue Bedienung des externen Programms ipacct, sei an dieser Stelle auf seine englischsprachige Hilfeseite verwiesen :

NAME

ipacct - control the per user ip accounting

SYNOPSIS

ipacct [-aChnrvVw] [-c user] [-d user] [-g user]
ipacct add addr [mask addr] [weight int]
ipacct add default [weight int]
ipacct del addr [mask addr]

DESCRIPTION

Ipacct prints the per user ip accounting data from the kernel and can also be used to control certain features of the accounting.

ipacct add addr [mask addr] [weight int]

In this form ipacct adds addr to the kernel weight table. Mask specifies the netmask to use for this address. The decision if an entry matches is made just like in the routing of ip packets. The third argument is the weight argument which specifies with how much bytes a byte sent or received to/from this address gets accounted in the statistics. It is possible to leave out both the mask and the weight argument. In this case a default weight of one is chosen and a reasonable default for the netmask, too: If the addresses ends in a non zero value, it is assumed to be a full host address. If not so, a default net mask is chosen which matches the class of the specified net. That is, for a class c net the mask is 255.255.255.0, for a class b net 255.255.0.0 and 255.0.0.0 of a class a net.

ipacct add default [weight int]

In this form ipacct changes the default weight used for all addresses that are not found in the weight table. If you leave out the weight argument a weight of one is assumed.

ipacct del addr [mask addr]

In this third and last form ipacct deletes the specified address from the weight table. If you leave out the weight argument a default is chosen for the netmask (see above).

OPTIONS

The options are as follows:

-a Read the weight table from the file /etc/ip_acct_user

Example:

```
# /etc/ip_acct_user
#
# format: <address> <addressmask> <weight>
127.0.0.1       255.255.255.255         0
192.0.2.100     255.255.255.255         0
```

-c user
Reset the sent/received counters for a specific user, numeric user ids and user names are allowed

-C Reset all sent/received counters

-d user
Disallow net access for this user (excluding access to hosts with a weight of zero)

-g user
Allow net access for this user (reverses -d user option)

-h Display a short usage description

-n Don't display any headers in the statistics

-r Display the statistics in raw format (like in /proc/net/ip_acct_user , /proc/net/ip_acct_weight respectively)

-v Display version of ipacct

-V Display version of the kernel driver that ipacct was compiled with

-w Display weight table instead of accounting table

EXAMPLES

ipacct add 127.0.0.1 weight 0
Exclude localhost (127.0.0.1) from the statistics, that is, every byte sent or received to/from localhost is accounted as zero bytes.

ipacct add 192.0.2.0 mask 255.255.255.0 weight 0
Exclude the class c net 192.0.2.0 from the statistics.

ipacct add 192.0.2.100 weight 1024
Every byte received or transmitted from/to host 192.0.2.100 is accounted as 1024 bytes in the statistics.

ipacct del 127.0.0.1
Remove this entry from the weight table.

ipacct del 192.0.2.0 mask 255.255.255.0
Remove this class c net from the table.

FILES

/proc/net/ip_acct_user /proc/net/ip_acct_weight
/etc/ip_acct_user

BUGS

This is more a caveat than a bug: Some of ipacct´s options (especially -d, -g, -c and -C) might interact with the ipquota(8) , ipstats(8) and ipacctd(8) programs, because ipquota and ipstats change the values which are also altered by the above mentioned options.

```
SEE ALSO
       ipacctd(8) ipquota(8) ipstats(8)

AUTHOR
       Lars Fenneberg <lf@elemental.net>.
       Send bugs/enhancements to <rsmeyers@khk.org>.
```

Hier noch einige Beispiele für mögliche Ausgaben von ipacct :

In der bereits oben betrachteten Benutzertabelle aus /proc/net/ip_acct_user ermittelt ipacct den Benutzernamen zur Nummer und zeigt ein Flag nur an, wenn es von `00` abweichen würde.

```
User       Sent       Received   Flags
root       173064     804972
wwwrun     148490     8269
nobody     3424       0
mheiden    83049      543782
```

Wesentlich mehr formatiert ipacct an der Gewichtungstabelle /proc/net/ip_acct_weight. In dieser Darstellung wird sofort klar, dass localhost (`127.0.0.1`) sinnvollerweise mit `0` gewichtet wird. Wie oben beschrieben, kann man hier verschiedene Gewichte ebenso für gesamte Subnetze vergeben, was hier für das Campus-Subnetz der Universität Kaiserslautern in Zeile 2 gesetzt ist.

```
Addr            Mask            Weight
127.0.0.1       255.255.255.255 0
131.246.166.54  255.255.0.0     0
default         *               1
```

Da wir uns zur Erfassung der Accountingdaten ohnehin schon an den richtigen Stellen des Sendens und Empfangens von Transfereinheiten im Kernel befinden, kann hier ohne weiteren großen Aufwand ein zusätzliches Feature angeboten werden. Wie der Manpage bei den Optionen -d und -g schon zu entnehmen ist, wird hier in der Benutzertabelle das zusätzliche Flag gesetzt, das im Kernel durch die ip_output.c ausgelesen wird und je nach Wert bestimmten Benutzern den Netzzugriff sperrt bzw. gewährt.

6. Ergebnisse und Zusammenfassung

Insgesamt kann man das Network-Accounting unter Linux als gelungen bewerten, wenn man die derzeitigen Möglichkeiten des aktuellen Kernels kennt und zugrundelegt.

Das Prozessmanagement des Linux-Kernels ist nicht nur im Hinblick auf das Network-Accounting, sondern im allgemeinen für alle Arten des benutzerbezogenen Accounting nicht ideal geeignet, da die Möglichkeit besteht, Kind-Prozessen einen anderen Besitzer zuzuordnen, als denjenigen, dem der Vater-Prozess gehört.

Von dieser Möglichkeit wird rege Gebrauch gemacht und zumeist ist der neu zugewiesene Besitzer der mit allen Systemverwaltungsrechten ausgestattete Benutzer "root". Zu Programmen, die solche Besitzeränderungen benutzen, gehören "ping" und "ssh" als Beispiele für netzwerknutzende Applikationen.

Dieser Umstand ist selbstverständlich für das Network-Accounting recht ärgerlich, allerdings gibt es eine Open-Source-Projektgruppe, die sich um die Verbesserung des Prozessmanagements im Linux-Kernel zur Ermöglichung einer besseren Abrechenbarkeit der Ressourcennutzung kümmert.

Wie in [1] auf den Seiten 26/27 bereits angesprochen, muss die Abrechnung eines Terminalservers als Ganzes gemessen werden und die Verteilung der Netzlast aufgrund von Accountingmechanismen der Server erfolgen. Nachdem eine Linux-Maschine ohne weiteres als Terminalserver eingesetzt werden kann, ist dieser Schritt mit der vorliegenden Arbeit vorgenommen, denn sie bietet ja gerade die Möglichkeit der benutzerbasierten Abrechnung der Netzlast in Umgebungen, in denen nicht jedem Benutzer eine eigene IP-Adresse eineindeutig zugewiesen werden kann.

7. Ausblick

Wie in Abschnitt 6 dargelegt, kann zumindestens für Linux die Hoffnung bestehen, dass der Kernel sein Prozessmanagement verbessert und somit als Socketeigentümer einer Datenübertragung immer der letztendliche Programmnutzer eingesetzt wird.

Für die mannigfaltigen Arten von Unix-Systemen wäre eine ähnliche Entwicklung notwendig, falls diese noch nicht vollzogen ist.

Die vorgestellte Software hat für den Fall der verbindungsorientierten Übertragung einen direkten Erweiterungspunkt, nämlich die Erkennung, welcher Kommunikationspartner die Verbindung zu Beginn bzw. vor der eigentlichen Übertragung aufgebaut hat. Hier ist es wieder eine politische bzw. wirtschaftswissenschaftliche Frage, wie dann der Server (Dienstanbieter) oder/und der Client (Dienstnutzer) letzten Endes zur Kasse gebeten werden, wobei derzeit sowohl beim Client als auch beim Server die volle Last der Datenübertragung erfasst wird.

8. Literatur

[1] Volker Bauer, Ralf Kleinfeld.
Entwurf und Implementierung eines Netzwerk-Accounting-Paketes NAccPak.

[2] Stefan Fischer, Walter Müller.
Netzwerkprogrammierung unter LINUX und UNIX.
Carl Hanser Verlag, 1996.
ISBN 3-446-18677-8.

[3] Andrew S. Tanenbaum.
Computer-Netzwerke.
Wolfram's Fachverlag, 1992, 2.Auflage.
ISBN 3-925328-79-3.

[4] Lehrbrief Informatik.
Einführung in die EDV.
ZFUW Kaiserslautern, 6.Auflage, 1999.

9. Anhang

9.1. Quellcode: Patch für Kernel 2.2.13

Verbale Kurzbeschreibung

```
--- linux/Documentation/Configure.help Wed Oct 20 02:14:00 1999
+++ linux-ipacct/Documentation/Configure.help    Sat Nov 13 19:50:26 1999
@@ -2412,6 +2412,15 @@

   If in doubt, say N here.

+IP: Per user accounting
+CONFIG_IP_ACCT_USER
+  This program allows you to account the IP traffic on a per user basis.
+  Usually you only want to say Y here if your box will be an IP provider
+  and you want to keep track of your users IP traffic.  This package does
+  not require you to use the default kernel IP accounting.  The data is
+  accessible with "cat /proc/net/ip_acct_user", so you want to say Y to
+  the /proc filesystem, if you say Y here.
+
 IP: firewall packet netlink device
 CONFIG_IP_FIREWALL_NETLINK
   If you say Y here, you can use the ipchains tool to copy all or part
```

Konfigurationsauswahl

```
--- linux/arch/i386/defconfig Thu Aug 26 02:29:46 1999
+++ linux-ipacct/arch/i386/defconfig    Sat Nov 13 19:50:26 1999
@@ -117,6 +117,7 @@
 # CONFIG_NET_IPGRE is not set
 # CONFIG_IP_ALIAS is not set
 # CONFIG_SYN_COOKIES is not set
+CONFIG_IP_ACCT_USER=y

 #
 # (it is safe to leave these untouched)
```

Vereinbarung von Datenstrukturen und Konstanten

```
--- /dev/null       Wed Mar  3 18:20:11 1999
+++ linux-ipacct/include/linux/ip_acct_user.h    Sat Nov 13 20:04:47 1999
@@ -0,0 +1,119 @@
+/*
+ * Copyright (C) 1994,1995 Lars Fenneberg
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU General Public License for more details.
+ *
+ * You should have received a copy of the GNU General Public License
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
+ *
+ */
+
+#ifndef _LINUX_IP_ACCT_USER_H
+#define _LINUX_IP_ACCT_USER_H
+
+#include <linux/types.h>
+
+#define IPAU_VERSION "IPAcctUser: version 0.7e\n"
+
+#define IPAU_NOUSER                    (uid_t)(65534)      /* thats the uid of nobody on my system */
+#define IPAU_DEF_WEIGHT                1         /* default weight */
+
+#define IPAU_HAS_RESET_BY_RW 1         /* can reset the user counters by open rw */
+#define IPAU_RESET_FREE                4         /* at the forth reset free the structure */
+
+/* see also linux/sockios.h for future collisions */
+
+#define SIOCADDWT  0x89D0              /* Add weight entry    */
+#define SIOCDELWT  0x89D1              /* Del weight entry    */
+#define SIOCRSTUT  0x89D2              /* Reset counter(s)    */
+#define SIOCCTRLU  0x89D3              /* Control user access */
+
+/* Magic value "Clear all acct info" for SIOCRSTUT ioctl */
+#define IPAU_CLEAR_ALL       -1
+
+/* Flags for SIOCCTRLU ioctl                                        */
+#define IPAU_NO_ACCESS        1
+
+/* structure for argument to SIOCCTRLU ioctl               */
+struct ip_acct_user_ctrl_entry {
+
+         uid_t     uid;
+         int       flags;
+
+};
```

```
+
+/* structure for argument to SIOCADDWT, SIOCDELWT ioctl  */
+struct ip_acct_weight_entry {
+
+         unsigned long addr;
+         unsigned long mask;
+         int       weight;
+
+};
+
+#ifdef __KERNEL__
+
+#define IPAU_HASHMAX                  101
+#define IPAU_HASHVAL(i)               ((i)%IPAU_HASHMAX)
+
+#ifdef DEBUG
+#define IPAU_MAGIC          0x4701
+#endif
+
+struct ip_acct_user_s {
+
+#ifdef DEBUG
+         int magic;
+#endif
+
+         uid_t     uid;      /* user who owns the socket     */
+         int       sent;     /* bytes sent for this user     */
+         int       recv;     /* bytes received for this user */
+         int       flags;  /* deny/grant access          */
+         char      reset;    /* counter for memory management*/
+
+         struct ip_acct_user_s *prev, *succ;
+
+};
+
+struct  ip_acct_weight_s {
+
+#ifdef DEBUG
+         int magic;
+#endif
+
+         unsigned long addr;
+         unsigned long mask;
+         int       weight;                     /* Weight of this address */
+                                               /* (0 means ignore)       */
+         struct ip_acct_weight_s *succ;
+
+};
+
+struct ip_acct_weight_list {
+
+         int default_weight;
+         struct ip_acct_weight_s *head;
+         struct ip_acct_weight_s *tail;
+
+};
+
+int ip_acct_user_sent(uid_t user, unsigned long daddr, int sent);
+int ip_acct_user_received(uid_t user, unsigned long daddr, int recv);
+int ip_acct_user_allowed(uid_t user, unsigned long daddr);
+int ip_acct_user_get_uinfo(char *buffer, char **start, off_t offset, int length, int reset);
+int ip_acct_user_get_winfo(char *buffer, char **start, off_t offset, int length, int unused);
+int ip_acct_user_ioctl(unsigned int cmd, void *arg);
+void ip_acct_user_init(void);
+
+#endif /* __KERNEL__ */
+#endif /* _LINUX_IP_ACCT_USER_H */
```

Vereinbarung der virtuellen Dateien im /proc-Dateisystem

```
--- linux/include/linux/proc_fs.h      Wed Oct 20 02:14:02 1999
+++ linux-ipacct/include/linux/proc_fs.h         Sat Nov 13 19:58:19 1999
@@ -146,6 +146,8 @@
          PROC_NET_IPFW_CHAIN_NAMES,
          PROC_NET_AT_AARP,
          PROC_NET_BRIDGE,
+         PROC_NET_IP_ACCT_USER,
+         PROC_NET_IP_ACCT_WEIGHT,
          PROC_NET_LAST
 };
```

Bool'sche Werte zur Voraus-wahl der Komponenten

```
--- linux/net/ipv4/Config.in  Wed Oct 20 02:14:02 1999
+++ linux-ipacct/net/ipv4/Config.in    Sat Nov 13 19:50:27 1999
@@ -53,6 +53,7 @@
       fi
     fi
 fi
+bool 'IP: Per user accounting' CONFIG_IP_ACCT_USER
 bool 'IP: optimize as router not host' CONFIG_IP_ROUTER
 tristate 'IP: tunneling' CONFIG_NET_IPIP
 tristate 'IP: GRE tunnels over IP' CONFIG_NET_IPGRE
```

Abfrage der Variablen, was kompiliert wird

```
--- linux/net/ipv4/Makefile   Tue Jan  5 00:31:35 1999
+++ linux-ipacct/net/ipv4/Makefile     Sat Nov 13 19:50:27 1999
@@ -105,6 +105,10 @@
 IPV4_OBJS += ipconfig.o
 endif

+ifdef CONFIG_IP_ACCT_USER
+IPV4_OBJS += ip_acct_user.o
+endif
+
 ifdef CONFIG_INET
 O_OBJS := $(IPV4_OBJS)
 OX_OBJS := $(IPV4X_OBJS)
```

Definition von Socketkommandos und deren Verhalten

```
--- linux/net/ipv4/af_inet.c  Mon Aug  9 21:05:13 1999
+++ linux-ipacct/net/ipv4/af_inet.c    Sat Nov 13 19:50:27 1999
@@ -100,6 +100,9 @@
 #include <net/ipip.h>
 #include <net/inet_common.h>
 #include <linux/ip_fw.h>
+#ifdef CONFIG_IP_ACCT_USER
+#include <linux/ip_acct_user.h>
+#endif
 #ifdef CONFIG_IP_MROUTE
 #include <linux/mroute.h>
 #endif
@@ -897,6 +900,13 @@
 #endif
                             if (rarp_ioctl_hook != NULL)
                                     return(rarp_ioctl_hook(cmd,(void *) arg));
+#ifdef CONFIG_IP_ACCT_USER
+                    case SIOCADDWT:
+                    case SIOCDELWT:
+                    case SIOCRSTUT:
+                    case SIOCCTRLU:
+                             return(ip_acct_user_ioctl(cmd,(void *)arg));
+#endif
                     case SIOCGIFADDR:
                     case SIOCSIFADDR:
                     case SIOCGIFBRDADDR:
@@ -1111,6 +1121,12 @@
           */

          icmp_init(&inet_family_ops);
+         /*
+          * Set the ICMP layer up
+          */
+#ifdef CONFIG_IP_ACCT_USER
+         ip_acct_user_init();
+#endif

          /* I wish inet_add_protocol had no constructor hook...
             I had to move IPIP from net/ipv4/protocol.c :-( --ANK
```

Bereitstellung aller Routinen für die Erfassung und Verwaltung der Accountingdaten

```
--- /dev/null       Wed Mar  3 18:20:11 1999
+++ linux-ipacct/net/ipv4/ip_acct_user.c         Sat Nov 13 19:50:27 1999
@@ -0,0 +1,667 @@
+/*
+ * Copyright (C) 1994,1995 Lars Fenneberg
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU General Public License for more details.
+ *
+ * You should have received a copy of the GNU General Public License
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/socket.h>
+#include <linux/kernel.h>
+#include <linux/string.h>
+
+#include <asm/segment.h>
+#include <linux/malloc.h>
+#include <linux/errno.h>
+#include <linux/types.h>
+#include <linux/in.h>
+#include <linux/inet.h>
+#include <linux/net.h>
+#include <linux/proc_fs.h>
+#include <linux/stat.h>
+
+#undef DEBUG
+#include <linux/ip_acct_user.h>
+
+#include <asm/uaccess.h>
+
+#define VERIFY_READ 0
```

```
+
+static struct ip_acct_user_s *user_stat[IPAU_HASHMAX];
+static struct ip_acct_weight_list user_weight_list;
+
+/*********************************************************************/
+/*********************************************************************/
+
+inline struct ip_acct_weight_s *
+match_addr(struct ip_acct_weight_list *wlist, unsigned long waddr)
+{
+  struct ip_acct_weight_s *p;
+
+  for (p = wlist->head; p != NULL; p = p->succ)
+  {
+    if (!(((p->addr) ^ waddr) & (p->mask))) break;
+  }
+
+#ifdef DEBUG
+  if ((p != NULL) && (p->magic != IPAU_MAGIC))
+  {
+    printk("ipacct: weight list corrupt.\n");
+    p = NULL;
+  }
+#endif
+
+  return p;
+}
+
+/*********************************************************************/
+
+static int
+add_addr(struct ip_acct_weight_list *wlist,
+                          struct ip_acct_weight_entry *wt)
+{
+  struct ip_acct_weight_s *p = wlist->head;
+
+#ifdef DEBUG
+  printk("ipacct: add addr.\n");
+#endif
+
+  if ((wt->addr) == 0x00000000) {
+    wlist->default_weight = wt->weight;
+    return 0;
+  }
+
+  cli();
+  while ((p != NULL) &&
+         (((p->addr) != (wt->addr)) ||
+         ((p->mask) != (wt->mask)))) p = p->succ;
+
+#ifdef DEBUG
+  if ((p != NULL) && (p->magic != IPAU_MAGIC))
+  {
+    printk("ipacct: weight list corrupt.\n");
+    p = NULL;
+  }
+#endif
+
+  if (p != NULL)
+  {
+    p->weight = wt->weight;
+  }
+  else
+  {
+    if ((p = (struct ip_acct_weight_s *)kmalloc(sizeof(struct ip_acct_weight_s),GFP_KERNEL)) == NULL) return -ENOMEM;
+
+#ifdef DEBUG
+    p->magic  = IPAU_MAGIC;
+#endif
+    p->addr   = wt->addr;
+    p->mask   = wt->mask;
+    p->weight = wt->weight;
+    p->succ   = NULL;
+
+    if (wlist->head == NULL)
+    {
+         wlist->head = wlist->tail = p;
+    }
+    else
+    {
+         wlist->tail->succ = p;
+         wlist->tail = p;
+    }
+
+  }
+  sti();
+
+  return 0;
+}
+
+/*********************************************************************/
+
+static int
+del_addr(struct ip_acct_weight_list *wlist,
+                          struct ip_acct_weight_entry *wt)
+{
+  struct ip_acct_weight_s *p = wlist->head;
+  struct ip_acct_weight_s *pred = NULL;
```

```
+
+
+#ifdef DEBUG
+  printk("ipacct: del addr.\n");
+#endif
+
+  cli();
+
+  while ((p != NULL) &&
+         (((p->addr) != (wt->addr)) ||
+         ((p->mask) != (wt->mask))))
+  {
+    pred = p;
+    p = p->succ;
+  }
+
+#ifdef DEBUG
+  if ((p != NULL) && (p->magic != IPAU_MAGIC))
+  {
+    printk("ipacct: weight list corrupt.\n");
+    p = NULL;
+  }
+#endif
+
+  if (p == NULL) return -EINVAL;
+
+  if (pred == NULL)
+  {
+    wlist->head = p->succ;
+    if (wlist->head == NULL) wlist->tail = NULL;
+  }
+  else
+  {
+    pred->succ = p->succ;
+    if (pred->succ == NULL) wlist->tail = pred;
+  }
+
+  sti();
+  kfree_s(p,sizeof(struct ip_acct_weight_s));
+
+  return 0;
+}
+
+/*********************************************************************/
+
+inline int
+get_weight(struct ip_acct_weight_list *wlist, unsigned long waddr)
+{
+  struct ip_acct_weight_s *p;
+  int w;
+
+  cli();
+  if ((p = match_addr(wlist, waddr)) == NULL) w = wlist->default_weight; else w = (p->weight);
+  sti();
+
+  return w;
+}
+
+/*********************************************************************/
+
+static int
+get_weight_info(struct ip_acct_weight_list *wlist, char *buffer,
+                          char **start, off_t offset, int length)
+{
+         struct ip_acct_weight_s *p = wlist->head;
+         int len=0;
+         off_t pos=0;
+         off_t begin=0;
+
+#ifdef DEBUG
+         printk("ipacct: get weight info.\n");
+#endif
+
+
+         len+=sprintf(buffer, "addr\t\tmask\t\tweight\n");
+
+         cli();
+         while(p != NULL)
+         {
+
+           len+=sprintf(buffer+len,"%08lX\t%08lX\t%-5i\n",p->addr,p->mask,p->weight);
+
+           p = p->succ;
+
+           pos=begin+len;
+           if(pos<offset)
+           {
+             len=0;
+             begin=pos;
+           }
+           if(pos>offset+length) break;
+         }
+         sti();
```

```
+
+         if (pos<=offset+length) {
+
+           len+=sprintf(buffer+len,"%08lX\t%08lX\t%-5i\n",htonl(0x00000000),htonl(0xffffffff),wlist->default_weight);
+
+           pos=begin+len;
+           if(pos<offset)
+           {
+             len=0;
+             begin=pos;
+           }
+
+         }
+
+         *start=buffer+(offset-begin);
+         len-=(offset-begin);
+         if(len>length)
+                  len=length;
+         return len;
+}
+
+/*****************************************************************/
+/*****************************************************************/
+
+static struct ip_acct_user_s *
+find_user(uid_t user)
+{
+  struct ip_acct_user_s *p = user_stat[IPAU_HASHVAL(user)];
+
+  while ((p != NULL) && ((p->uid) != user)) p = p->succ;
+
+#ifdef DEBUG
+  if ((p != NULL) && (p->magic != IPAU_MAGIC))
+  {
+    printk("ipacct: user list corrupt.\n");
+    p = NULL;
+  }
+#endif
+
+  return p;
+}
+
+/*****************************************************************/
+
+static struct ip_acct_user_s *create_user(uid_t user)
+{
+    struct ip_acct_user_s *p;
+    int hashval = IPAU_HASHVAL(user);
+
+#ifdef DEBUG
+    printk("ipacct: create user (uid %i).\n",user);
+#endif
+
+    if ((p = (struct ip_acct_user_s *)kmalloc(sizeof(struct ip_acct_user_s),GFP_ATOMIC)) == NULL)
+         return NULL;
+
+#ifdef DEBUG
+    p->magic  = IPAU_MAGIC;
+#endif
+    p->uid    = user;
+    p->reset  = 0;
+    p->prev   = NULL;
+    p->succ   = user_stat[hashval];
+    if (p->succ) p->succ->prev = p;
+    user_stat[hashval] = p;
+
+    return p;
+}
+
+/*****************************************************************/
+
+void delete_user(struct ip_acct_user_s *p)
+{
+         int hashval = IPAU_HASHVAL(p->uid);
+
+         if (p->prev)
+         {
+                  p->prev->succ = p->succ;
+                  if (p->succ) p->succ->prev = p->prev;
+         }
+         else
+         {
+                  user_stat[hashval] = p->succ;
+                  if (user_stat[hashval]) user_stat[hashval]->prev = NULL;
+         }
+         kfree_s(p, sizeof(struct ip_acct_user_s));
+}
+
+/*****************************************************************/
+
+static int
+ctrl_access(struct ip_acct_user_ctrl_entry *ct)
+{
+  struct ip_acct_user_s *p;
+
+#ifdef DEBUG
+  printk("ipacct: ctrl access (uid %i, flags %x).\n",ct->uid, ct->flags);
+#endif
```

```
+
+  if (ct->uid == 0)
+         return -EPERM;               /* even root cannot change the flags */
+                                      /* for the root account                   */
+
+  cli();
+  if ((p = find_user(ct->uid)))
+  {
+    p->flags = ct->flags;
+  }
+  else if (ct->flags)
+  {
+    if ((p = create_user(ct->uid)) == NULL) return -ENOMEM;
+
+    p->sent   = 0;
+    p->recv   = 0;
+    p->flags  = ct->flags;
+
+  }
+  sti();
+
+  return 0;
+}
+
+/*********************************************************************/
+
+static int
+reset_counter(int uid)
+{
+  struct ip_acct_user_s *p, *succ;
+  int i;
+  int result = 0;
+
+#ifdef DEBUG
+  printk("ipacct: reset counter (uid %i).\n",uid);
+#endif
+
+  if (uid>=0)
+  {
+    cli();
+    if ((p = find_user((uid_t)uid)))
+    {
+      if ((p->reset >= IPAU_RESET_FREE) && !(p->flags))
+      {
+         delete_user(p);
+      }
+      else
+      {
+        if (p->sent + p->recv)
+        {
+                  p->sent = p->recv = p->reset = 0;
+         }
+         else
+         {
+                  p->reset++;
+         }
+      }
+
+    } else result = -EINVAL;
+    sti();
+  }
+  else
+  {
+    cli();
+    for(i=0;i<IPAU_HASHMAX; i++)
+    {
+       p = user_stat[i];
+       while (p)
+       {
+          succ = p->succ;
+          if ((p->reset >= IPAU_RESET_FREE) && !(p->flags))
+          {
+
+              delete_user(p);
+          }
+          else
+          {
+             if (p->sent + p->recv)
+             {
+                   p->sent = p->recv = p->reset = 0;
+              }
+              else
+              {
+                   p->reset++;
+              }
+          }
+          p = succ;
+       }
+    }
+    sti();
+  }
+  return result;
+}
+
+/*********************************************************************/
```

```
+
+int
+ip_acct_user_allowed(uid_t user, unsigned long daddr)
+{
+  struct ip_acct_user_s *p;
+  unsigned long flags;
+
+  save_flags(flags);
+  cli();
+  if ((p = find_user(user)))
+  {
+    if ((p->flags & IPAU_NO_ACCESS) && get_weight(&user_weight_list, daddr))
+    {
+      restore_flags(flags);
+      return 0;
+    }
+  }
+  restore_flags(flags);
+  return 1;
+}
+
+/*****************************************************************/
+
+int
+ip_acct_user_sent(uid_t user,unsigned long daddr, int sent)
+{
+  struct ip_acct_user_s *p;
+  unsigned long flags;
+
+#ifdef DEBUG
+  printk("ipacct: user %i daddr %s sent %i\n",user,in_ntoa(daddr),sent);
+#endif
+
+  save_flags(flags);
+  cli();
+  if ((p = find_user(user)))
+  {
+
+    p->sent += sent*get_weight(&user_weight_list, daddr);
+
+  }
+  else
+  {
+    if ((p = create_user(user)) == NULL) return -ENOMEM;
+
+    p->sent   = sent*get_weight(&user_weight_list, daddr);
+    p->recv   = 0;
+    p->flags  = 0;
+
+  }
+  restore_flags(flags);
+
+  return 0;
+}
+
+int
+ip_acct_user_received(uid_t user, unsigned long daddr, int recv)
+{
+  struct ip_acct_user_s *p;
+  unsigned long flags;
+
+#ifdef DEBUG
+  printk("ipacct: user %i daddr %s recv %i\n",user,in_ntoa(daddr),recv);
+#endif
+
+  save_flags(flags);
+  cli();
+  if ((p = find_user(user)))
+  {
+
+    p->recv += recv*get_weight(&user_weight_list, daddr);
+
+  }
+  else
+  {
+    if ((p = create_user(user)) == NULL) return -ENOMEM;
+
+    p->sent   = 0;
+    p->recv   = recv*get_weight(&user_weight_list, daddr);
+    p->flags  = 0;
+
+  }
+  restore_flags(flags);
+
+  return 0;
+}
+
+/*****************************************************************/
+
+int
+ip_acct_user_get_uinfo(char *buffer, char **start, off_t offset, int length, int reset)
+{
+         struct ip_acct_user_s *p, *succ;
+         int i;
+         int len=0;
+         int last_len=0;
+         off_t pos=0;
+         off_t begin=0;
+         unsigned long flags;
```

```
+
+#ifdef DEBUG
+         printk("ipacct: get user info.\n");
+#endif
+
+         len+=sprintf(buffer, "uid      sent       recv       flags\n");
+
+
+         for(i = 0; i < IPAU_HASHMAX; i++)
+         {
+                   save_flags(flags);
+                   cli();
+
+                   p = user_stat[i];
+                   while(p != NULL)
+                   {
+                             succ = p->succ;
+
+                             len+=sprintf(buffer+len,"%-8i %-10i %-10i %02X\n",p->uid,p->sent,p->recv,p->flags);
+
+                             pos=begin+len;
+                             if(pos<offset)
+                             {
+                                       len=0;
+                                       begin=pos;
+                             }
+                             else if(pos>offset+length)
+                             {
+                                       len = last_len;
+                                       break;
+                             }
+                             else if(reset)
+                             {
+                                       if ((p->reset >= IPAU_RESET_FREE) && !(p->flags))
+                                       {
+                                                 delete_user(p);
+                                       }
+                                       else
+                                       {
+                                                 if (p->sent + p->recv)
+                                                 {
+                                                           p->sent = p->recv = p->reset = 0;
+                                                 }
+                                                 else
+                                                 {
+                                                           p->reset++;
+                                                 }
+                                       }
+                             }
+
+                             last_len = len;
+                             p = succ;
+                   }
+
+                   restore_flags(flags);
+
+                   if(pos>offset+length)
+                             break;
+         }
+         *start=buffer+(offset-begin);
+         len-=(offset-begin);
+         if(len>length)
+                   len=length;
+         return len;
+}
+
+/*****************************************************************************/
+
+int
+ip_acct_user_get_winfo(char *buffer, char **start, off_t offset, int length, int unused)
+{
+         return get_weight_info(&user_weight_list, buffer, start, offset, length);
+}
+
+/*****************************************************************************/
+
+void
+ip_acct_user_init(void)
+{
+         int i;
+
+         proc_net_register(&(struct proc_dir_entry) {
+                   PROC_NET_IP_ACCT_USER, 12, "ip_acct_user",
+                   S_IFREG | S_IRUGO | S_IWUSR, 1, 0, 0,
+                   0, &proc_net_inode_operations,
+                   ip_acct_user_get_uinfo
+         });
+
+         proc_net_register(&(struct proc_dir_entry) {
+                   PROC_NET_IP_ACCT_WEIGHT, 14, "ip_acct_weight",
+                   S_IFREG | S_IRUGO | S_IWUSR, 1, 0, 0,
+                   0, &proc_net_inode_operations,
+                   ip_acct_user_get_winfo
+         });
+
+         printk(IPAU_VERSION);
+
+         for(i=0; i<IPAU_HASHMAX; i++) user_stat[i] = NULL;
+
+         user_weight_list.head = user_weight_list.tail = NULL;
+         user_weight_list.default_weight = IPAU_DEF_WEIGHT;
+
+}
```

```
+
+/*****************************************************************/
+
+int ip_acct_user_ioctl(unsigned int cmd, void *arg)
+{
+         int err;
+         int uid;
+         struct ip_acct_user_ctrl_entry ct;
+         struct ip_acct_weight_entry wt;
+
+         switch(cmd)
+         {
+                  case SIOCADDWT:              /* Add a weight */
+                  case SIOCDELWT:              /* Delete a weight */
+                           if (!suser())
+                                    return -EPERM;
+                           err=!access_ok(VERIFY_READ, arg, sizeof(struct ip_acct_weight_entry));
+                           if (err)
+                                    return err;
+                           copy_from_user(&wt, arg, sizeof(struct ip_acct_weight_entry));
+                           return (cmd == SIOCADDWT) ? add_addr(&user_weight_list, &wt) : del_addr(&user_weight_list, &wt);
+                  case SIOCRSTUT:              /* Reset user table */
+                           if (!suser())
+                                    return -EPERM;
+                           err=!access_ok(VERIFY_READ, arg, sizeof(int));
+                           if (err)
+                                    return err;
+                           copy_from_user(&uid, arg, sizeof(int));
+                           return (reset_counter(uid));
+                  case SIOCCTRLU:              /* Ctrl user access */
+                           if (!suser())
+                                    return -EPERM;
+                           err=!access_ok(VERIFY_READ, arg, sizeof(struct ip_acct_user_ctrl_entry));
+                           if (err)
+                                    return err;
+                           copy_from_user(&ct, arg, sizeof(struct ip_acct_user_ctrl_entry));
+                           return (ctrl_access(&ct));
+         }
+
+         return -EINVAL;
+}
```

Abfrage des Berechtigungsflags und Aufruf der Routine zum Schreiben der Accountingdaten

```
--- linux/net/ipv4/ip_output.c         Wed Oct 20 02:14:02 1999
+++ linux-ipacct/net/ipv4/ip_output.c  Sat Nov 13 19:24:30 1999
@@ -76,6 +76,10 @@
 #include <linux/firewall.h>
 #include <linux/mroute.h>
 #include <linux/netlink.h>
+#ifdef CONFIG_IP_ACCT_USER
+#include <linux/ip_acct_user.h>
+#endif
+

 /*
  *      Shall we try to damage output packets if routing dev changes?
@@ -266,6 +270,14 @@
          if(opt && opt->is_strictroute && rt->rt_dst != rt->rt_gateway)
                   goto no_route;

+#ifdef CONFIG_IP_ACCT_USER
+        /*
+         * IS the user allowed to send
+         */
+        if ((skb->sk) && (skb->sk->socket) && !ip_acct_user_allowed(SOCK_INODE(skb->sk->socket)->i_uid, sk->daddr))
+                return (-EPERM);
+#endif
+
          /* We have a route, so grab a reference. */
          skb->dst = dst_clone(sk->dst_cache);

@@ -292,6 +304,15 @@
          iph->id = htons(ip_id_count++);

          dev = rt->u.dst.dev;
+
+#ifdef CONFIG_IP_ACCT_USER
+        if (sk && sk->socket) {
+          ip_acct_user_sent(SOCK_INODE(sk->socket)->i_uid, iph->daddr,ntohs(iph->tot_len));
+        }
+        else {
+          ip_acct_user_sent(IPAU_NOUSER, iph->daddr, ntohs(iph->tot_len));
+        }
+#endif

 #ifdef CONFIG_FIREWALL
          /* Now we have no better mechanism to notify about error. */
```

Aufruf der Routine zum Schreiben der Accountinginformationen

```
--- linux/net/ipv4/raw.c      Mon Aug  9 21:04:41 1999
+++ linux-ipacct/net/ipv4/raw.c        Sat Nov 13 19:26:31 1999
@@ -61,6 +61,10 @@
 #include <net/udp.h>
 #include <net/raw.h>
 #include <net/checksum.h>
+#ifdef CONFIG_IP_ACCT_USER
+#include <linux/ip_acct_user.h>
+#endif
+
 #ifdef CONFIG_IP_MROUTE
 struct sock *mroute_socket=NULL;
@@ -171,6 +175,10 @@
                  kfree_skb(skb);
                  return -1;
         }
+
+#ifdef CONFIG_IP_ACCT_USER
+ if (sk->socket) ip_acct_user_received(SOCK_INODE(skb->sk->socket)->i_uid, skb->nh.iph->saddr, ntohs(skb->nh.iph->tot_len));
+#endif

         ip_statistics.IpInDelivers++;
         return 0;
```

Aufruf der Routine zum Schreiben der Accountinginformationen

```
--- linux/net/ipv4/tcp_ipv4.c Wed Oct 20 02:14:02 1999
+++ linux-ipacct/net/ipv4/tcp_ipv4.c    Sat Nov 13 19:29:28 1999
@@ -63,6 +63,10 @@
 #include <linux/inet.h>
 #include <linux/stddef.h>

+#ifdef CONFIG_IP_ACCT_USER
+#include <linux/ip_acct_user.h>
+#endif
+
 extern int sysctl_tcp_timestamps;
 extern int sysctl_tcp_window_scaling;
 extern int sysctl_tcp_sack;
@@ -1760,6 +1764,11 @@

         if (sk->state == TCP_TIME_WAIT)
                  goto do_time_wait;
+
+#ifdef CONFIG_IP_ACCT_USER
+ if (sk->socket) ip_acct_user_received(SOCK_INODE(sk->socket)->i_uid, skb->nh.iph->saddr, ntohs(skb->nh.iph->tot_len));
+#endif
+
         if (!atomic_read(&sk->sock_readers))
                  return tcp_v4_do_rcv(sk, skb);
```

Aufruf der Routine zum Schreiben der Accountinginformationen

```
--- linux/net/ipv4/udp.c      Mon Aug  9 21:05:10 1999
+++ linux-ipacct/net/ipv4/udp.c        Sat Nov 13 19:31:24 1999
@@ -114,6 +114,9 @@
 #include <net/icmp.h>
 #include <net/route.h>
 #include <net/checksum.h>
+#ifdef CONFIG_IP_ACCT_USER
+#include <linux/ip_acct_user.h>
+#endif

 /*
  *      Snmp MIB for the UDP layer
@@ -1125,6 +1128,7 @@
 #endif
         sk = udp_v4_lookup(saddr, uh->source, daddr, uh->dest, skb->dev->ifindex);

+
         if (sk == NULL) {
 #ifdef CONFIG_UDP_DELAY_CSUM
                  if (skb->ip_summed != CHECKSUM_UNNECESSARY &&
@@ -1141,6 +1145,12 @@
                  kfree_skb(skb);
                  return(0);
         }
+
+#ifdef CONFIG_IP_ACCT_USER
+        if (sk && sk->socket)
+          ip_acct_user_received(SOCK_INODE(sk->socket)->i_uid, saddr, ntohs(skb->nh.iph->tot_len));
+#endif
+
         udp_deliver(sk, skb);
         return 0;
```

9.2. Quellcode: Externes Steuermodul

```
/*
 * Copyright (C) 1994,1995 Lars Fenneberg
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 */

/* Parts extracted from Net-3, Authors Fred N. van Kempen, Alan Cox,
 * and from route.c, Authors Fred N. van Kempen, Johannes Stille, Linus
 * Torvalds, Alan Cox.
 */

#include <sys/types.h>
#include <sys/ioctl.h>
#include <ctype.h>
#include <pwd.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <netinet/in.h>

#include "pathnames.h"
#include <linux/ip_acct_user.h>

#define DENY 1
#define ALLOW 0

char pname[255];
int raw = 0;
int disp_weight = 0;
int no_header = 0;
int skfd = -1;

void usage(void)
{
  fprintf(stderr,"Usage: %s [-wvVrahCn] [-c <user>] [-d <user>] [-g <user>]\n",pname);
  fprintf(stderr,"       %s add <addr> [mask <addr>] [weight <int>]\n",pname);
  fprintf(stderr,"       %s add default [weight <int>]\n",pname);
  fprintf(stderr,"       %s del <addr> [mask <addr>]\n",pname);
  exit(1);
}

void version(void)
{
  fprintf(stderr,"%s: version %s\n",pname, VERSION);
  exit(1);
}

void drv_version(void)
{
  fprintf(stderr,"%s: compiled for kernel driver: %s", pname, IPAU_VERSION);
  exit(1);
}

void error(char *str)
{
  fprintf(stderr,"%s: %s.\n",pname, str);
  exit(1);
}

int is_addr(char *str)
{
  while(isdigit(*str)) str++;
  if (*(str++) != '.') return 0;
  while(isdigit(*str)) str++;
  if (*(str++) != '.') return 0;
  while(isdigit(*str)) str++;
  if (*(str++) != '.') return 0;
  while(isdigit(*str)) str++;
  if (!isdigit(*str) && (*str !=  '\0')) return 0;
  return 1;
}

/*
 *        Display an IP address in readable format.
 */
```

```
char *in_ntoa(unsigned long in)
{
          static char buff[18];
          char *p;

          p = (char *) &in;
          sprintf(buff, "%d.%d.%d.%d",
                    (p[0] & 255), (p[1] & 255), (p[2] & 255), (p[3] & 255));
          return(buff);
}


/*
 *        Convert an ASCII string to binary IP.
 */

unsigned long in_aton(char *str)
{
          unsigned long l;
          unsigned int val;
          int i;

          l = 0;
          for (i = 0; i < 4; i++)
          {
                    l <<= 8;
                    if (*str != '\0')
                    {
                              val = 0;
                              while (*str != '\0' && *str != '.')
                              {
                                        val *= 10;
                                        val += *str - '0';
                                        str++;
                              }
                              l |= val;
                              if (*str != '\0')
                                        str++;
                    }
          }
          return(htonl(l));
}


unsigned long in_xton(char *str)
{

          char *sp = str;
          char *ptr;
          unsigned long saddr;
          int i,val;

          ptr = (unsigned char *) (&(saddr) + 1);
          for (i = 0; i < sizeof(saddr); i++) {
                    val = 0;
                    if (*sp == '\t')
                              break;
                    if (*sp >= 'A')
                              val = (int) (*sp - 'A') + 10;
                    else
                              val = (int) (*sp - '0');
                    val <<= 4;
                    sp++;
                    if (*sp >= 'A')
                              val |= (int) (*sp - 'A') + 10;
                    else
                              val |= (int) (*sp - '0');
                    *--ptr = (unsigned char) (val & 0377);
                    sp++;
          }
          return saddr;
}

unsigned long default_mask(unsigned long dst)
{
          dst = ntohl(dst);
          if ((dst & 0xff) != 0) return htonl(0xffffffff);
          if (IN_CLASSA(dst))
                    return htonl(IN_CLASSA_NET);
          if (IN_CLASSB(dst))
                    return htonl(IN_CLASSB_NET);
          return htonl(IN_CLASSC_NET);
}
```

```
int reset_counter(char *user)
{
          int uid;
          struct passwd *pw;

          if (user && ((pw = getpwnam(user)) || (pw = getpwuid(atoi(user)))))
          {
                    uid = pw->pw_uid;
          }
          else
          {
            if (user)
            {
              fprintf(stderr,"%s: unknown user %s.\n",pname, user);
              return (1);
            }
            else
            {
              uid = IPAU_CLEAR_ALL;
            }
          }

          if (ioctl(skfd, SIOCRSTUT, &uid) < 0) {
                    fprintf(stderr, "%s: SIOCRSTUT: %s.\n",pname, strerror(errno));
                    return (1);
          }
          return (0);
}

int ctrl_access(char *user, int deny)
{
          struct ip_acct_user_ctrl_entry ct;
          struct passwd *pw;

          if ((pw = getpwnam(user)) || (pw = getpwuid(atoi(user))))
          {
                    ct.uid = pw->pw_uid;
                    ct.flags = deny?IPAU_NO_ACCESS:0;
          }
          else
          {
                    fprintf(stderr,"%s: unknown user %s.\n",pname, user);
                   return (1);
          }

          if (ioctl(skfd, SIOCCTRLU, &ct) < 0) {
                    fprintf(stderr, "%s: SIOCCTRLU: %s.\n",pname, strerror(errno));
                    return (1);
          }
          return (0);
}

int weight_add(unsigned long waddr, unsigned long wmask, int weight)
{
          struct ip_acct_weight_entry wt;

          wt.addr = waddr;
          wt.mask = wmask;
          wt.weight = weight;

          if (ioctl(skfd, SIOCADDWT, &wt) < 0) {
                    fprintf(stderr, "%s: SIOCADDWT: %s.\n",pname, strerror(errno));
                    return (1);
          }
          return (0);
}

int weight_del(unsigned long waddr, unsigned long wmask)
{
          struct ip_acct_weight_entry wt;

          wt.addr = waddr;
          wt.mask = wmask;

          if (ioctl(skfd, SIOCDELWT, &wt) < 0) {
                    fprintf(stderr, "%s: SIOCDELWT: %s.\n",pname, strerror(errno));
                    return (1);
          }
          return (0);
}
```

```
void read_table(void)
{
          FILE *table;
          char waddr[18], wmask[18], buff[1024];
          int weight;
          int line = 0;

          if ((table = fopen(_PATH_ETC_IP_ACCT_USER,"r")) == NULL)
          {
            perror(_PATH_ETC_IP_ACCT_USER);
          }

          while (fgets(buff, 1023, table))
          {
            line ++;
            if (buff[0] == '#') continue;

            if (sscanf(buff, "%17s %17s %i\n",waddr, wmask, &weight)==3)
            {
              if (!is_addr(waddr) || !is_addr(wmask))
              {
                fprintf(stderr, "%s: wrong address format (line %i).\n", pname, line);
              }
              if (weight<0)
              {
                fprintf(stderr, "%s: weight < 0 (line %i).\n", pname, line);
              }
              weight_add(in_aton(waddr),in_aton(wmask),weight);
            }
            else fprintf(stderr, "%s: syntax error (line %i).\n", pname, line);
          }

          fclose(table);
          exit(0);
}

void print_wt(void)
{

  char buff[1024], waddr[18], wmask[18];
  int weight,num;
  FILE *fp;

  if ((fp = fopen(_PATH_PROCNET_IP_ACCT_WEIGHT, "r")) == NULL)
  {
     perror(_PATH_PROCNET_IP_ACCT_WEIGHT);
     return;
  }

   if (!no_header) printf("Addr            Mask            Weight\n");

   while (fgets(buff,1023,fp))
   {

     num = sscanf(buff, "%s %s %i\n",waddr,wmask,&weight);
     if (num != 3) continue;

     if (!raw)
     {
       if (in_xton(waddr)==htonl(0x00000000))
       {
           strcpy(waddr,"default");
           strcpy(wmask,"*");
       }
       else
       {
         strcpy(waddr,in_ntoa(in_xton(waddr)));
         strcpy(wmask,in_ntoa(in_xton(wmask)));
       }
     }
     printf("%-15s %-15s %-3i\n",waddr,wmask,weight);
   }
   fclose(fp);
}
```

```
void print_ut(void)
{

  char buff[1024], sflags[5];
  int sent, recv, num, uid, flags;
  FILE *fp;
  struct passwd *pw;

  if ((fp = fopen(_PATH_PROCNET_IP_ACCT_USER, "r")) == NULL)
  {
     perror(_PATH_PROCNET_IP_ACCT_USER);
     return;
  }

   if (!no_header) printf("User       Sent       Received   Flags\n");

   while (fgets(buff,1023,fp))
   {

     num = sscanf(buff, "%i %i %i %x\n",&uid,&sent,&recv,&flags);
     if (num != 4)
     {
       num = sscanf(buff, "%i %i %i\n",&uid,&sent,&recv);
       if (num!=3) continue;
       flags = 0;
     }

     if (sent == 0 && recv == 0 && flags == 0) continue;

     *sflags = '\0';
     strcat(sflags, (flags & IPAU_NO_ACCESS)?"N":"");

     if (raw || (pw = getpwuid((uid_t)uid)) == NULL)
     {
        printf("%-10i %-10i %-10i %-4s\n",uid,sent,recv,sflags);
     }
     else
     {
        printf("%-10s %-10i %-10i %-4s\n",pw->pw_name,sent,recv,sflags);
     }
   }
   fclose(fp);
}

int main(int argc, char **argv)
{
  extern int optind;
  extern char *optarg;
  int c, n, nmax, def, add = 0;
  int weight = IPAU_DEF_WEIGHT;
  unsigned long waddr = 0;
  unsigned long wmask = 0;

  if (strrchr(argv[0],'/') == NULL)
  {
    strcpy(pname,argv[0]);
  }
  else
  {
    strcpy(pname,strrchr(argv[0],'/')+1);
  }

  if ((skfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
  {
    fprintf(stderr,"%s: can't create socket (%s)\n.",pname, strerror(errno));
    return 1;
  }

  while ((c = getopt(argc,argv,"wvVrahc:Cnd:g:")) > 0)
  {
          switch (c)
          {
                    case 'w':
                                        disp_weight++;
                                        break;
                    case 'v':
                                        version();
                                        break;
                    case 'V':
                                        drv_version();
                                        break;
                    case 'r':
                                        raw++;
                                        break;
                    case 'a':
                                        read_table();
                                        break;
                    case 'c':
                                        return (reset_counter(optarg));
                    case 'C':
                                        return (reset_counter(NULL));
                    case 'd':
                                        return (ctrl_access(optarg,DENY));
                    case 'g':
                                        return (ctrl_access(optarg,ALLOW));
                    case 'n':
                                        no_header++;
                                        break;
```

```
                  case 'h':
                  default:
                                    usage();
                                    break; /* never reached */
          }
  }

  n = optind;
  nmax = argc-1;

  if (n>nmax)
  {
    if (disp_weight) print_wt(); else print_ut();
  }
  else
  {
    if (!strcasecmp(argv[n],"add"))
    {
      add = 1;
    }
    else if (!strcasecmp(argv[n],"del"))
    {
      add = 0;
    }
    else error("wrong command (use 'add' or 'del')");

    n++;
    if (n>nmax) error("address missing");

    if (add && !strcasecmp(argv[n],"default"))
    {
      waddr = htonl(0x00000000);
      wmask = htonl(0xffffffff);
      def = 1;
    }
    else
    {
      if (!is_addr(argv[n])) error("address has wrong format");
      waddr = in_aton(argv[n]);
      def = 0;
    }

    n++;
    while (n<=nmax)
    {
       if (!def && !strcasecmp(argv[n],"mask"))
       {
           n++; if (n>nmax) error("attribute 'mask' requires an argument");
           if (!is_addr(argv[n])) error("attribute 'mask': address has wrong format");
           wmask = in_aton(argv[n]);
       }
       else if (add && !strcasecmp(argv[n],"weight"))
       {
           n++; if (n>nmax) error("attribute 'weight' requires an argument");
           weight = atoi(argv[n]);
       } else error((add)?((def)?"wrong attribute (use 'weight')":"wrong attribute (use 'mask' and 'weight')"):
                          "wrong attribute (use 'mask')");
       n++;
    }

    if (wmask==0) wmask = default_mask(waddr);
    if (add) return weight_add(waddr, wmask, weight); else return weight_del(waddr,wmask);

  }

  return 0;
}
```