

# Routing mit QoS-Eigenschaften unter Linux

Falko Dressler, Ursula Hilgers

Regionales Rechenzentrum der Universität Erlangen–Nürnberg, Martensstrasse 1  
Falko.Dressler@rrze.uni-erlangen.de / Ursula.Hilgers@rrze.uni-erlangen.de

**Zusammenfassung** Durch die zunehmende Anzahl interaktiver Multimedia-Anwendungen steigen die Anforderungen an die Netzwerk-Funktionalität. Die Notwendigkeit, Dienstgüte für die Übertragung zuzusichern, gewinnt zunehmend an Bedeutung. In dieser Arbeit soll die Traffic Control Funktionalität des Linux Betriebssystems präsentiert werden. Dabei werden zunächst die von der IETF standardisierten Dienstgüte-Mechanismen für IP-Netzwerke vorgestellt. Die Implementation unter Linux wird erläutert und anschließend in Labormessungen untersucht.

## 1 Einleitung und Motivation

In vielen Bereichen der Wissenschaft und Industrie gehören multimediale Datenübertragungen bereits zum Standardumfang des Dienstleistungsangebotes. Auch viele kleinere Forschungsinstitute und Firmen sind dabei, sich Dienste wie Voice- und Video-Übertragungen über die gleiche Netzinfrastruktur nutzbar zu machen, über die Daten übertragen werden.

Linux ist ein mittlerweile weltweit akzeptiertes Betriebssystem, welches immer weitere Verbreitung findet. Aufgrund des niedrigen Preises (das System an sich ist kostenfrei) und der relativ einfachen Bedienung ist dieses System bereits heute erste Wahl für die Realisierung des Routers für den Anschluss ans Internet gerade auch für kleine Forschungsinstitute und Firmen [Rie99]. Interessant wird diese Konstruktion auch deshalb, da der Linux-Router auch als Firewall genutzt werden kann. In dieser Arbeit sollen die Traffic Control Funktionen untersucht werden, um die der Linux-Kernel erweitert wird [Lor00]. Diese sollen in der Lage sein, Dienstgüte in Netzwerken bereitzustellen.

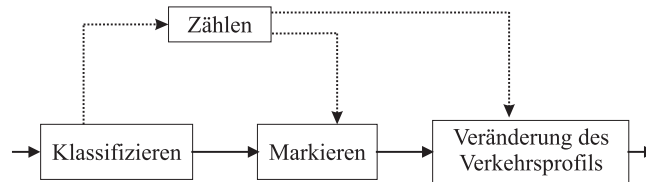
Im zweiten Kapitel werden zunächst von der IETF standardisierte Dienstgüte-Mechanismen für IP-Netzwerke vorgestellt. Im dritten Kapitel wird die Implementation unter Linux erläutert und anschließend in Labormessungen untersucht.

## 2 QoS für IP

Um das IP-Protokoll um die Fähigkeit zu erweitern, Dienstgüte in Netzwerken bereitzustellen, hat die IETF zwei unterschiedliche Ansätze vor-

geschlagen. Die Integrated Services Architektur [Bra94], reserviert für jeden Anwendungsfluss Ressourcen in den Endgeräten und den Routern im Netzwerk. Ein Einsatz dieses Verfahrens scheidet zur Zeit noch an der Komplexität und der schlechten Skalierbarkeit des zugrundeliegenden Reservierungsprotokolls RSVP [Bra97], [Man97].

Ein anderer Ansatz ist das Differentiated Services Konzept der IETF [Bla98], das alle IP-Pakete in wenige Dienstklassen mit unterschiedlichen Dienstcharakteristiken einordnet. Dazu werden IP-Pakete klassifiziert und zu Dienstklassen mit unterschiedlichen Qualitätsmerkmalen aggregiert. Die Zuordnung eines Paketes zu einer Dienstklasse erfolgt durch einen Wert im IP-Header, dem Differentiated Services-Byte (DiffServ-Byte) [Nic98].



**Abbildung1.** Komponenten der DiffServ-Architektur [Bla98].

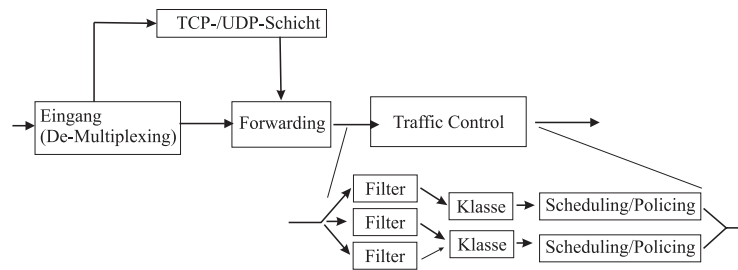
Pakete, die in einen Router gelangen, werden vom Klassifizierer ausgewertet und der Wert des DiffServ-Bytes bestimmt die Dienstklasse, der das Paket angehört (Abbildung 1). Nach der Klassifikation überprüft ein Zähler, ob das Verkehrsprofil der Pakete einer Dienstklasse, beispielsweise die gesendete Bandbreite, den vertraglichen Vereinbarungen entspricht. Dieses Ergebnis entscheidet, ob ein Markierer den Wert im DiffServ-Byte des IP-Headers mit einem anderen Wert belegt, der das Paket möglicherweise auf Grund von zu hohem Bandbreitenbedarf in eine niedrigere Dienstklasse einstuft. Darüber hinaus besteht die Möglichkeit, Pakete wegen Verletzung der vertraglich ausgehandelten Parameter bei Überschreiten der zugestandenen Bandbreite zu verwerfen oder zu verzögern, um den Verkehrsstrom zu glätten.

Für jede Dienstklasse und dem entsprechenden Wert im DiffServ-Byte wird durch ein Per Hop Behaviour (PHB) festgelegt, wie Pakete im ausgehenden Interface des Routers weitergeleitet werden. Das PHB wird z. B. durch Scheduling-Verfahren oder durch Mechanismen zur Verhinderung

von Überlast in Netzwerken realisiert. Außerdem werden in Abhängigkeit von der Priorität der Dienstklasse Ressourcen reserviert.

### 3 Implementation unter Linux

Der Linux Kernel erlaubt die Unterstützung der meisten Funktionalitäten des DiffServ-Konzeptes [WWW00]. Die Implementation basiert auf einem modularen Ansatz, der aus den folgenden Komponenten besteht: Den Filtern, den Klassen, den Scheduling-Verfahren und dem Policing / Shaping.



**Abbildung 2.** DiffServ-Komponenten des Linux-Kernels [Alm99].

In Abbildung 2 ist dargestellt, wie der Linux-Kernel Daten, die er vom Netzwerk empfängt, im Eingangsinterface verarbeitet und sie den zugehörigen Applikationen zuleitet. Nicht in die Skizze integriert sind Funktionen, die bereits am Eingangsinterface die IP-Pakete untersuchen und aufgrund der gewählten Konfiguration Pakete verwerfen bzw. ein (Re-)Tagging vornehmen. Diese Funktionen werden in der vorliegenden Arbeit nicht untersucht. Die Komponente **Forwarding** empfängt Informationen aus den höheren Schichten und führt alle Aufgaben zur korrekten Weiterleitung der Daten durch, wie z. B. die Wahl des Ausgangsinterfaces. Bevor die Daten auf das Netzwerk weitergeleitet werden, wird auf sie die Traffic Control Funktionalität angewendet. **Filter** werden dazu verwendet, Pakete in verschiedene **Klassen** mit unterschiedlichen Leistungscharakteristiken einzuordnen. Beispielsweise kann Netzwerkkontrollverkehr, der an eine bestimmte Portnummer gesendet wird, aus dem übrigen Datenverkehr herausgefiltert werden, um ihn bevorzugt zu behandeln. In dem Beispiel in Abbildung 2 teilen drei verschiedene Filter den Verkehr auf zwei Klassen auf. Mit jeder Netzwerkschnittstelle ist

ein **Scheduling**-Verfahren gekoppelt, das festlegt, in welcher Reihenfolge Daten gesendet werden. Es können mehrere Warteschlangen verwaltet werden (s. Abbildung 2). Zusätzlich kann mit Hilfe von **Policing** Verkehr von Klassen, die die ihnen zugestandene Bandbreite überschreiten, verworfen werden. Somit kontrollieren Scheduling-Verfahren mit Policing-Funktionalität das Puffermanagement am ausgehenden Interface eines Routers. In der Linux-Implementation sind mehrere Scheduling-Verfahren realisiert, von denen zwei miteinander verglichen werden sollen: Die Standard-Methode, **FIFO-Queueing** (First in, First Out), speichert Pakete in einer Warteschlange in der Reihenfolge, mit der sie im System eintreffen, und leitet sie dann weiter. **Class Based Queueing** (CBQ) wird in [FJ95] als **Queue-Management-Verfahren** vorgestellt, in den Messungen aber als Scheduling-Verfahren angewendet. Es teilt den Verkehr der verschiedenen Klassen in eine hierarchische Struktur. Dabei wird jeder Klasse der ihr zustehende Anteil an Ausgangsbandbreite zugewiesen. Dadurch wird die Nutzung einer physikalischen Verbindung durch mehrere logisch getrennte Datenflüsse ermöglicht. Zusätzlich unterstützt das Verfahren durch die Isolierung und die mögliche Bevorzugung von Datenverkehr die Übertragung von Echtzeit-Daten.

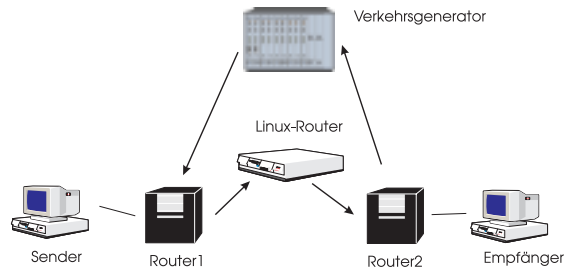
## 4 Laboruntersuchungen

Mit den folgenden Tests soll die praktische Nutzbarkeit von Linux-Systemen als integrierte Routing-Komponente mit Traffic Control Funktionalität zur Bereitstellung von QoS untersucht werden. Neben der Klassifizierung, die die Grundlage für unterschiedliche Behandlung von Verkehr ist, soll anhand verschiedener Fallstudien das Verhalten eines Linux-Routers untersucht werden, wenn verschiedene Datentypen wie z.B. Telefonie über IP (VoIP) oder auch Videoübertragungen über dieselbe Netzinfrastruktur übertragen werden.

### 4.1 Testaufbau

Die vorgestellten Tests werden mit dem in Abbildung 3 dargestellten Messaufbau durchgeführt. Der Linux-Router ist mit zwei Ethernetkarten ausgestattet, eine mit einer Bandbreite von 10 MBit/s und eine mit 100 MBit/s. Ein Verkehrsgenerator der Firma Smartbits (SMB6000) erzeugt UDP-Verkehr. Da leider nur PoS-Interfaces (Packet over Sonet) am Smartbits zur Verfügung stehen, wird Router der Firma Cisco (Cisco 7500) zur Umsetzung von PoS bzw. FDDI auf Ethernet eingesetzt. Die

100 MBit/s Ethernetkarte des Linux-Routers ist mit **Router1** verbunden. Weiterhin kommen zwei UNIX-Systeme der Firma Sun (Sun ULTRA 60) zum Einsatz, um den TCP-Verkehr zu erzeugen. Sie sind mit den Cisco-Routern über FDDI angeschlossen. Eine Referenzmessung zeigt, daß an den Eingangs- und Ausgangsschnittstellen dieser Router keine Pakete verworfen werden.



**Abbildung3.** Testaufbau für die Labormessungen.

Im folgenden wird zunächst die Filter- und Klassifikationsfunktionalität in der Linux-Realisierung untersucht. Danach werden FIFO und CBQ mit sich unterscheidenden Verkehrsszenarien verglichen.

## 4.2 Klassifizierung von Verkehr

Der Linux-Scheduler ist in der Lage, die Klassifizierung der IP-Pakete anhand verschiedener Merkmale durchzuführen. Zum einen kann das ToS-Byte (Type of Service) ausgewertet werden. Aber auch eine Klassifizierung der IP-Pakete anhand von Adressen im IP-Header ist möglich. In den Tests wird die Unterscheidung von Verkehrsströmen anhand der Ziel-IP-Adressen durchgeführt. Zum Einsatz kam dabei die Firewall-Funktion des Linux-Routers. Ein spezieller Teil der Firewall-Regeln dient dazu, IP-Pakete zu identifizieren (anhand von IP-Adresse und Port) und verschiedenen Klassen zuzuordnen. Das Kommando `ipchains` ordnet einen Verkehrsstrom in eine Klasse ein und weist ihm ein Label zu. In dem unten stehenden Beispiel werden zwei verschiedene ausgehende Flows (Datenströme mit identischem Quadrupel [Quell-IP-Adresse, Quell-IP-Port, Ziel-IP-Adresse, Ziel-IP-Port]) anhand der Zieladressen zwei Klassen mit den Labels 2 und 3 zugeordnet:

```
ipchains -A output -d 192.129.1.0/24 -m 2
ipchains -A output -d 192.129.2.0/24 -m 3
```

Das folgende Beispiel veranschaulicht, wie die Markierung mit Hilfe von Filterregeln über das ToS-Byte erfolgen kann. Mit der Maske `0xfc` werden die ersten 14 Bits aus diesem Byte maskiert. Mit den unten abgebildeten Beispielen werden alle Pakete mit dem Wert 0 im ToS-Byte in die Klasse mit dem Label 2 einsortiert und die mit dem Wert `0x20` in die Klasse mit Label 3.

```
tc filter add dev eth0 parent 1:0 protocol ip prio 100 u32 match ip precedence \
0x00 0xfc flowid 1:2
tc filter add dev eth0 parent 1:0 protocol ip prio 100 u32 match ip precedence \
0x20 0xfc flowid 1:3
```

### 4.3 Vergleich der Scheduling-Mechanismen: reiner UDP-Verkehr

Um die beiden Scheduling-Verfahren FIFO-Scheduling und CBQ miteinander zu vergleichen, werden vier Verkehrsflüsse mit folgenden Eigenschaften definiert. Angegeben sind die Senderate des Verkehrsgenerators, der Typ des Datenverkehrs, der simuliert werden soll und die konfigurierte Rate der Dienstklassen bei den CBQ-Tests. **Flow1** soll die höchste Priorität haben, **Flow4** die niedrigste.

**Flow1:** Senderate: 33 Frames/s á 240 Bytes, entspricht 63 kBit/s  
Typ: hochpriorer Sprachverkehr (VoIP)  
max. CBQ-Kanalrate: 100 kBit/s

**Flow2:** Senderate: 33 Frames/s á 240 Bytes, entspricht 63 kBit/s  
Typ: Multimedia, Audio  
max. CBQ-Kanalrate: 100 kBit/s

**Flow3:** Senderate: 139 Frames/s á 900 Bytes, entspricht 1 MBit/s  
Typ: Multimedia, Video  
max. CBQ-Kanalrate: 1.1 MBit/s

**Flow4:** Senderate: 2331-14568 Frames/s á 429 Bytes,  
entspricht 8 MBit/s-50 MBit/s  
Typ: "Best Effort" Verkehr  
max. CBQ-Kanalrate: 8 MBit/s

Im ersten Versuch wird mit dem Standard-FIFO-Scheduler des Routers gearbeitet, um eine Referenz für den folgenden CBQ-Test zu erhalten. Dabei wird die Senderate auf **Flow4** kontinuierlich erhöht. Untersucht werden sollten die Auswirkungen auf den Durchsatz von **Flow1**

bis **Flow3**. In Tabelle 1 sind die Ergebnisse zusammengestellt. Erkennbar ist, daß bei steigender Senderate auf **Flow4** alle anderen Flows deutliche Durchsatzeinbußen erkennen lassen. Der Verkehr von **Flow1** – **Flow3** wird verdrängt durch die große Senderate auf **Flow4**.

**Tabelle1.** Durchsatz bei FIFO-Scheduling mit 4 UDP-Flows.

Senderate	Durchsatz			
Flow4	Flow1	Flow2	Flow3	Flow4
[ <i>MBit/s</i> ]	[ <i>kBit/s</i> ]	[ <i>kBit/s</i> ]	[ <i>kBit/s</i> ]	[ <i>kBit/s</i> ]
8.00	63.22	63.22	991.36	7976.08
10.00	54.67	52.51	825.2	8434.83
12.00	48.16	44.71	707.72	8598.25
20.01	31.97	26.38	422.28	8873.49
49.97	16.69	10.43	197.32	9194.71

In einem zweiten Test wird das Verhalten der Verkehrsströme unter Einsatz des CBQ-Scheduler untersucht. Dazu werden vier Klassen erzeugt und diese anhand der IP-Adressen unterschiedlichen Klassen zugeordnet. Die Rate der einzelnen CBQ-Kanäle ist, wie oben beschrieben konfiguriert.

In Tabelle 2 sind die Ergebnisse zusammengestellt. Man sieht, daß die Datenströme **Flow1** – **Flow3** keine Datenverluste erleiden müssen. Der Durchsatz bleibt konstant, obwohl die Senderate von **Flow4** wie im Test vorher kontinuierlich erhöht wird.

**Tabelle2.** Durchsatz bei CBQ-Scheduling mit 4 UDP-Flows.

Senderate	Durchsatz			
Flow4	Flow1	Flow2	Flow3	Flow4
[ <i>MBit/s</i> ]	[ <i>kBit/s</i> ]	[ <i>kBit/s</i> ]	[ <i>kBit/s</i> ]	[ <i>kBit/s</i> ]
8.00	63.43	63.43	994.72	8003.35
10.00	63.46	63.46	995.4	8184.35
12.00	62.54	62.54	980.52	8066.34
20.01	62.37	62.37	977.96	8043.79
49.97	62.42	62.42	978.6	8051.95

In Abbildung 4 ist der Vergleich zwischen FIFO- und CBQ-Scheduling zusammenfassend dargestellt. Wie bereits beschrieben, reduziert sich bei

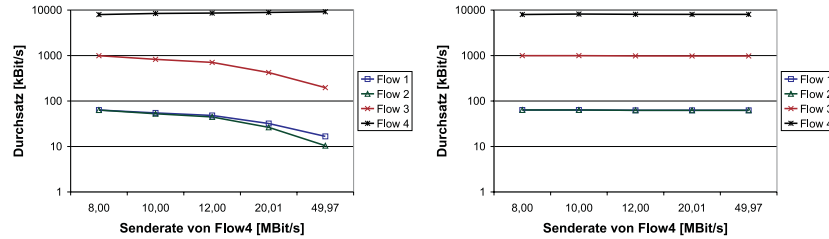


Abbildung 4. Messungen mit reinem UDP-Verkehr (links FIFO, rechts CBQ).

FIFO-Scheduling der Durchsatz von Flow 1 – Flow 3 bei steigender Senderate von Flow 4, während bei CBQ der Durchsatz dieser höherpriorigen Flows nicht reduziert wird, auch wenn die Überlast am ausgehenden Interface von Router 2 steigt.

#### 4.4 Vergleich der Scheduling-Mechanismen: gemischter UDP- / TCP-Verkehr

In diesem Kapitel wird untersucht, ob TCP-Ströme mit dem TCP eigenen Congestion Control Mechanismen durch Anwendung von CBQ Scheduling-Strategien gegenüber einem "Best Effort"-Strom geschützt werden. Es werden drei verschiedene Flows konfiguriert mit folgenden Eigenschaften:

**Flow 1:** Senderate: 33 Frames/s á 240 Bytes, entspricht 63 kBit/s

Typ: hochpriorer Sprachverkehr (VoIP)

max. CBQ-Kanalrate: 100 kBit/s

**Flow 2:** Senderate: maximaler TCP-Durchsatz bei 1500 Bytes/Frame

Typ: TCP-Verkehr (ftp)

max. CBQ-Kanalrate: 4 MBit/s

**Flow 3:** Senderate: 2331–14568 Frames/s á 429 Bytes

entspricht 8 MBit/s–50 MBit/s

Typ: "Best Effort" Verkehr

max. CBQ-Kanalrate: 5 MBit/s

Wieder wird ein erster Versuch mit dem Standard-FIFO-Scheduler des Linux-Routers durchgeführt, um eine Referenz für den folgenden CBQ-Test zu erhalten. Die Senderate von Flow 3 wird kontinuierlich erhöht. In Tabelle 3 sind die Ergebnisse zusammengestellt. Es ist der gleiche Effekt zu erkennen wie in Tabelle 1. Der Durchsatz der hochpriorigen Klassen Flow 1 und Flow 2 sinkt mit steigender Senderate auf Flow 3.

**Tabelle3.** Durchsatz bei FIFO-Scheduling mit 2 UDP-Flows und einem TCP-Flow.

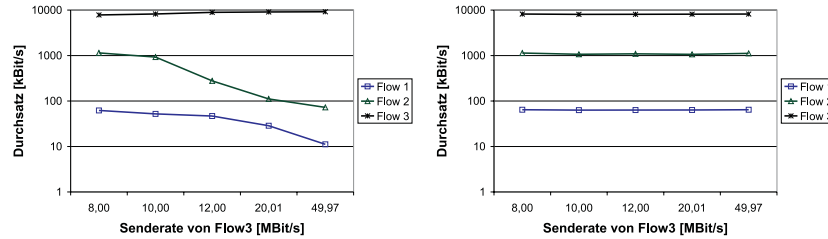
Senderate Flow3 [MBit/s]	Durchsatz		
	Flow1 [kBit/s]	Flow2 [kBit/s]	Flow3 [kBit/s]
8.00	62.11	1144.08	7817.07
10.00	52.19	926.56	8198.82
12.00	46.69	277.76	8911.07
20.01	28.61	110.96	9102.06
49.97	11.17	72.32	9166.24

Im nächsten, wesentlich interessanteren Test wird das Verhalten der Verkehrsströme unter Einsatz des CBQ untersucht. Dazu werden drei Klassen erzeugt und die anhand der IP-Adressen unterschiedenen IP-Ströme den Klassen zugeordnet.

In der Zusammenstellung der Ergebnisse in Tabelle 4 sieht man sehr gut, daß wieder der kleine hochpriore UDP-Strom (z.B. VoIP) ohne Datenverlust arbeiten kann. Etwas anders schaut das Bild beim TCP-Strom aus. Eine wesentliche Eigenschaft von TCP ist es, durch den sogenannten "Slow Start"-Mechanismus die gesendete Datenrate zu reduzieren. Immer, wenn Pakete verloren gehen, also durch TCP neu übertragen werden müssen, setzt dieser Mechanismus ein, der zunächst die Senderate reduziert, damit sich die vermeintliche Überlastsituation am Router entschärft, um die Rate dann wieder kontinuierlich zu steigern. Das Ergebniss ist, daß der TCP-Strom nicht, wie vermutet, volle 4 MBit/s übertragen kann, sondern nur ca. 1 MBit/s nutzt. Der "freie" Raum wird wieder durch die "Best Effort"-Klasse geborgt, so daß hier höhere Datenraten möglich sind.

**Tabelle4.** Durchsatz bei CBQ-Scheduling mit 2 UDP-Flows und einem TCP-Flow.

Senderate Flow3 [MBit/s]	Durchsatz		
	Flow1 [kBit/s]	Flow2 [kBit/s]	Flow3 [kBit/s]
8.00	64.32	1133.84	8167.53
10.00	63.04	1067.44	8054.33
12.00	63.33	1091.92	8079.73
20.01	63.49	1061.68.6	8119.25
49.97	64.32	116.80	8192.98



**Abbildung 5.** Messungen mit gemischtem UDP- und TCP-Verkehr (links FIFO, rechts CBQ).

Wie in Abbildung 5 zu sehen ist, bricht die Übertragungsleistung des TCP-Stroms bei höherer Last in der "Best Effort"-Klasse nicht ein, wenn CBQ aktiviert ist. Eine sinnvolle Aufteilung der zur Verfügung stehenden Bandbreite ist mittels CBQ möglich, aber durch die Strukturierung der Klassen anhand von festen IP-Adressen sehr statisch. Durch die Einteilung der Klassen anhand des gesetzeten ToS-Bytes, ist dies zwar etwas dynamischer, aber nur sehr wenige Applikationen und Endgeräte unterstützen dies bisher.

## 5 Zusammenfassung

Die Untersuchungen in dieser Arbeit zeigen, daß die Erweiterungen des Linux-Kernels um die Traffic Control Funktionalität zu zufriedenstellenden Ergebnissen führen. Die Labormessungen zeigen, daß die Klassifikation von Verkehr als Grundlage zur differentiellen Behandlung von IP-Paketen nach unterschiedlichen Kriterien möglich ist. Das implementierte Class Based Queueing ermöglicht eine Isolierung von hoch priorisierten Strömen und bietet auch in Überlastsituationen einen garantierten Durchsatz.

Einige vorgesehene Tests waren leider aufgrund der nicht vollständigen Implementierung nicht möglich. So konnte z.B. das Priority-Queueing nicht in die Labormessungen einbezogen werden.

Zusammenfassend kann man aber sagen, daß die Linux-Routing-Maschine für kleinere Forschungsinstitute und Firmen als kostengünstige Alternative zu z.B. Cisco-Routern empfohlen werden kann. Unter Linux sind die QoS-Features für alle Interfacekarten einsetzbar.

## Literatur

- [Alm99] W. Almesberger, *Linux Network Traffic Control - Implementation Overview*. <http://icawww1.epfl.ch/linux-diffserv/>, April, 1999.
- [Bla98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, *An Architecture for Differentiated Services*. Request for Comments 2475, December 1998.
- [Bra94] B. Braden, D. Clark, S. Shenker, *Integrated Services in the Internet Architecture: an Overview*. Request for Comments 1633, Juni 1994.
- [Bra97] B. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, *Resource ReSerVation Protocol (RSVP) — Version 1 Functional Specification*. Request for Comments 2205, September 1997.
- [FJ95] S. Floyd, V. Jacobson, *Link-sharing and Resource Management Models for Packet Networks*. In: IEEE/ACM Transactions on Networking, August 1995.
- [Lor00] M. Lorenz, *Geordnete Wege, Traffic Control mit Linux*. iX 4/2000.
- [Man97] A. Mankin, F. Baker, B. Braden, S. Bradner, M. O'Dell, A. Romanow, A. Weinrib, L. Zhang, *Resource ReSerVation Protocol (RSVP) — Version 1 Applicability Statement — Some Guidelines on Deployment*. Request for Comments 2208, September 1997.
- [Nic98] K. Nichols, S. Blake, F. Baker, D. L. Black, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. Request for Comments 2474, Dezember 1998.
- [Rie99] M. Riepe, *Spur Pinguin, Routing im Linux-Kernel 2.2.x*. iX 9/1999.
- [WWW00] *Differentiated Services on Linux*. <http://icawww1.epfl.ch/linux-diffserv/>, Juli 2000.