Heinlein & Partner
LINUX CONSULTING

Die nachfolgende Projektarbeit
wurde im Rahmen eines Praktikums bei
Heinlein & Partner - Linux Consulting
angefertigt.

http://www.heinlein-partner.de

Und wurde mit der Note 1 bewertet.
Wir gratulieren Michael Bonin!

Intelligentes Trafficmanagement für Internet Service Provider

Quality of Service mit Linux

Projektarbeit von
Michael Bonin

Staatliche Technikerschule Berlin

Fachbereich: Staatlich geprüfter Industrietechnologe
Fachrichtung: Datentechnik / Wirtschaft
Semester: ITD4

Betreuer: Herr Heinlein (JPBerlin)
Frau Gracklauer (Artplan21 GmbH)
Herr Reinecke (STB)

Berlin, 11.07.2003

Bewertung: _____

Inhaltsverzeichnis

1 Einleitung

Mein Praktikum brachte mich zur JPBerlin - Mailbox und Politischer Provider, einem Internet Service Provider, wo ich mit den allgemeinen Tätigkeiten eines Providers, sowie seinen Problemen, in Berührung kam. Mit der Firma BerliNet, ebenfalls ein Internet Service Provider, teilt sich die JPBerlin Technik. Beide stellen diverse Internetdienste zu Verfügung, wie z.B. Web, News, Email und Mailinglisten.

Die JPBerlin verfügt über eine 2,3 MBit S-DSL-Leitung.

Die Transfergeschwindigkeit steht für die JPBerlin im Vordergrund, Problem ist jedoch, dass sie nur über 2,3 MBit verfügt, da Bandbreite teuer ist. Sie versucht durch Optimierung gute und schnelle Datentransfers zu ermöglichen. Ein Upgrade auf eine schneller Leitung möchte die JPBerlin erst so spät wie möglich in Betracht ziehen.

Die S-DSL-Leitung kostet pauschal ca. 990 Euro pro Monat und ist volumenfrei („Flatrate“), d.h. dass die gesendete und empfangene Datenmenge nicht ausgewertet und zusätzlich bezahlt werden muss.

Trotz nicht vorhandener GigaByte-Preise hat die transportierbare Datenmenge einen monetären Wert. In der folgenden Rechnung wird das ein wenig deutlicher:

Teilt man die monatlichen Kosten von 990 Euro durch die maximal transportierbare Datenmenge pro Monat, dann erhält man die Kosten pro GigaByte.

In dem Beispiel der JPBerlin:

2,3 MBit / Sekunde = 0,2875 MB / Sekunde = 17,25 MB / Minute
= 1.035 MB / Stunde = 24.840 MB / Tag
= 745.200 MB / Monat (30 Tage)
= maximal 727,73 GB / Monat

990 Euro / 727,73 GB = 1,36 Euro/GB

Diese Rechnung ist natürlich nur fiktiv, da eine 100%ige Auslastung einer Leitung durchgehend 24 Stunden am Tag völlig unpraktikabel ist und somit auszuschließen ist. Die Datenverbindung wäre unbenutzbar.

Eine Leitung kann durchschnittlich bis ca. 30% ihrer Leistungsfähigkeit ausgelastet werden, bevor sie deutliche Geschwindigkeitseinbußen zeigt. Bei einer solchen Auslastung der Leitung ergibt sich, laut der vorangegangenen Rechnung, ein ungefährer Wert von 218,32 GByte / Monat und ein GigaByte-Preis von ca. 4,53 Euro / GByte.

Durch ein funktionierendes Traffic Shaping bzw. Bandbreitenmanagement kann die Leitung bis zu ca. 40% ausgelastet werden, bevor die Geschwindigkeitseinbußen spürbar werden. Ein Webhoster kann so bei gleichen Fixkosten durchschnittlich um ¼ mehr Traffic über die identische Leitung transportieren. Zum einem ist eine enorme Steigerung der Leitungsgeschwindigkeit spürbar, zum anderen kann er mehr Webspace an mehr Kunden verkaufen, bei gleichen Fixkosten.

Auch wenn Bandbreitenmanagement die Datenmenge nicht reduzieren kann, werden doch betriebswirtschaftlich Kosten gespart.

Ein Breitenmanagement bzw. ein Traffic Shaping einzuführen wird für einen Internet Service Provider und Webhoster immer bedeutsamer, weil der normale Internetbenutzer einen immer schnelleren Zugang hat. Zu „Modemzeiten“ war Traffic Shaping für einen Provider bei weitem noch nicht so interessant, wie es heute der Fall ist, wo Internetnutzer über Zugänge, wie T-DSL, mit einer Download-Bandbreite von ca. 768 kBit verfügen. Theoretisch können mittlerweile drei DSL-Nutzer die komplette Bandbreite der 2,3 MBit Leitung belegen, d.h. greifen mehrere DSL-Nutzer gleichzeitig auf den Provider zu, ist nicht mehr genug Bandbreite vorhanden, um alle Dienste, sowie Anfragen der Internetnutzer, mit Maximal-Geschwindigkeit zu versorgen.

Diese Arbeit dokumentiert und analysiert die Möglichkeiten die Traffic-Shaping bietet und zeigt auf, wie Provider durch relativ einfache und billige Maßnahmen ihre Ressourcen und Qualität deutlich verbessern können.

2 Ist-Analyse

Die Firmen JPBerlin und BerliNet sind Internet Service Provider und teilen sich eine gemeinsame S-DSL-Leitung mit einer maximalen Bandbreite von 2,3 MBit mit der die gemeinsamen Serverräume mit einem Pool von ca. 25 Servern angebunden sind. Beide Firmen teilen sich außerdem einen Primär-Multiplex-Anschluss (PriMux) mit 30 B-Kanälen für die Einwahl von Firmenkunden in das Internet.

2.1 Topologie

Alle Server und Router der beiden Provider basieren auf Linux in der Distribution SuSE in unterschiedlichen Versionen, auf Arbeitsplatzrechnern ist ebenfalls Linux vorzufinden, bis auf einige wenige Rechner. Das verwendete Betriebssystem von gehosteten Servern sowie Arbeitsplatzrechner der mit angebundenen Firma ist unbekannt.

In der folgenden Abbildung wird die Netzwerktopologie der beiden Internet Service Provider dargestellt. Diese Darstellung der Topologie entspricht nahezu der Realität, ist jedoch aus Sicherheitsgründen leicht verändert. An einigen Stellen fehlen daher Rechnernamen und IP-Adressen.

BerliNet und eine weitere Firma, die über die 2,3 MBit Leitung mit angebunden ist, um das Internet nutzen zu können, möchten keine Details ihres Netzwerkes bekannt geben.

Auch die JPBerlin möchte nicht alles Preis geben. In der Netzwerknachbildung der JPBerlin fehlen diverse Sicherheitseinrichtungen, wie Firewalls, Honeypots und Intrusion Detection Systeme.

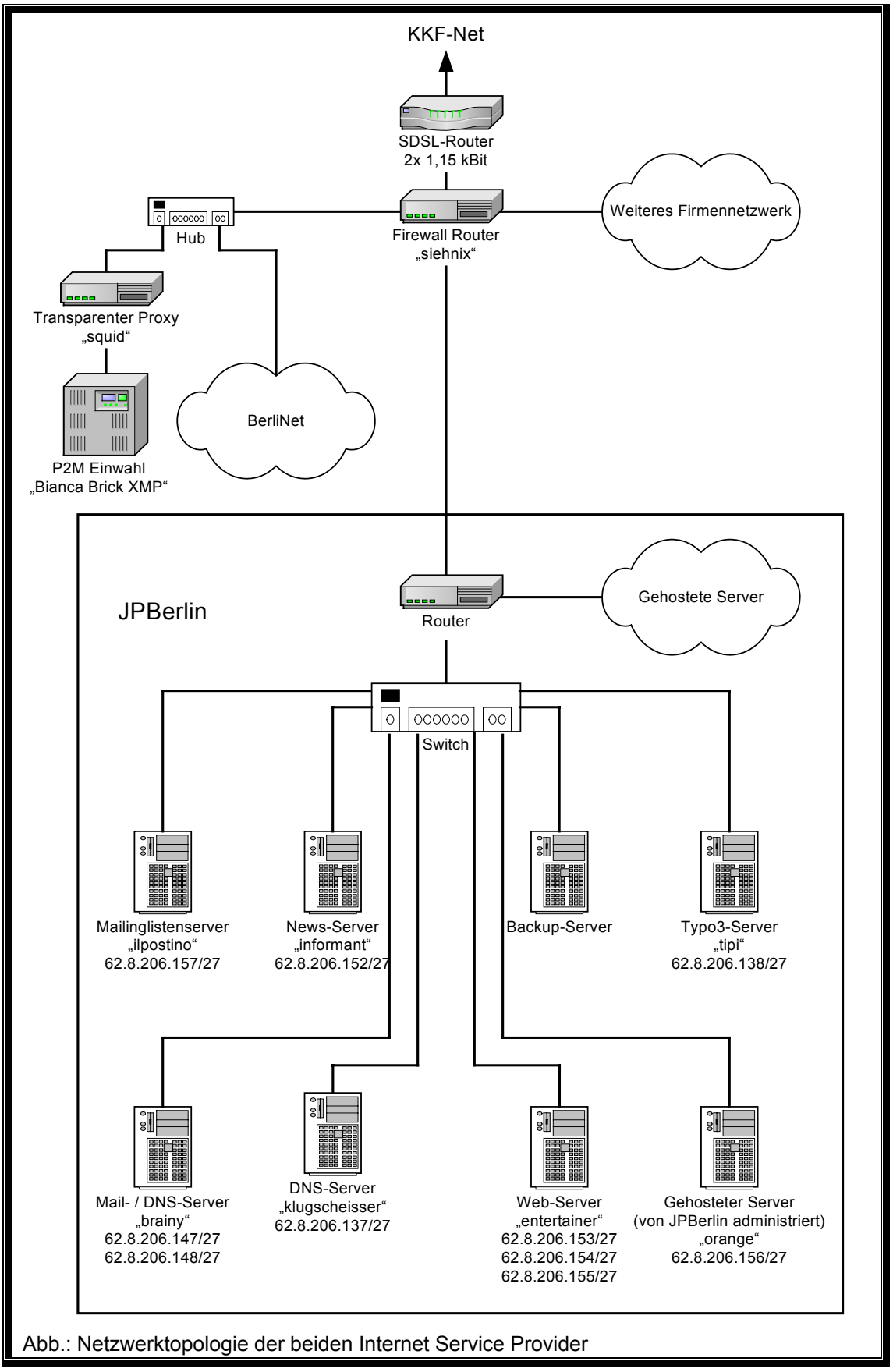

Abb.: Netzwerktopologie der beiden Internet Service Provider

2.2 Datenflussanalyse

Durch die zahlreichen Internet-Dienste, die beide Firmen anbieten, kommt es zu vielen gleichzeitigen Datenübertragungen, wobei es durchaus passieren kann, dass die komplette Bandbreite der 2,3 MBit S-DSL-Leitung bereits durch nur einen Dienst, wie z.B. durch die Abarbeitung von einer Mailingliste, belegt wird. Problematisch ist das, wenn Anfragen von normalen Webnutzern langsamer abgearbeitet werden, interaktive Sitzungen mit spürbaren Verzögerungen nutzbar und Down- bzw. Uploads verlangsamt werden.

2.2.1 „Aktiver“ und „passiver“ Traffic

Der vorhandene Datenverkehr ist aus Sicht der JPBerlin in zwei Gruppen differenzierbar, in „aktiven“ und „passiven“ Traffic.

„Aktiver Traffic“ ist für die JPBerlin als Webhoster weitaus wichtiger bzw. hochprioritär einzustufen. Zum aktiven Traffic zählen vor allem interaktive Verbindungen, wie SSH-Sitzungen, und direkte Verbindungen mit Internetnutzern, deren Anfragen möglichst schnell befriedigt werden sollen, dazu gehören z.B. Webaufrufe und das Empfangen von Emails (SMTP) und der Versand (POP3 / IMAP).

„Passiver Traffic“ ist weniger wichtig, dieser Traffic kann durchaus langsamer ablaufen. Dazu gehören Server-zu-Server-Verbindungen, bei denen größere Datenmengen übertragen werden. Beispiel hierfür ist die „Unterhaltung“ des Email-Servers der JPBerlin mit einem Email-Server von GMX, dabei wird meistens eine große Anzahl von Emails übertragen. Dieser Traffic kann als zweitrangig betrachtet werden, da keine ungeduldigen Nutzer warten. Daher können diese Übertragungen mit geringerer Bandbreite erfolgen.

2.2.2 Mailinglistenserver

Die Firma JPBerlin besitzt neben Email-Servern auch noch einen Mailinglistenserver. Allein durch diesen Mailinglistenserver „ilpostino.jpberlin.de“ kann bereits die gesamte Bandbreite der Leitung belegt werden. Dieser Mailinglistenserver schreibt gleichzeitig an eine große Anzahl von Email-Adressen. Dabei kann die Menge der angeschriebenen Adressen, bei nur einer eingehenden Email, problemlos eine Anzahl von bis zu 25.000 Email-Adressen erreichen.

Diese große Anzahl von Emails versucht der Server möglichst schnell zu versenden, indem er parallel mit bis zu fünfzehn Verbindungen sendet, d.h. er überträgt gleichzeitig an fünfzehn Email-Server, die über eine große Bandbreite verfügen und somit die gesamte Bandbreite der JPBerlin belegt ist.

Dieser Datenverkehr ist in die Gruppe des passiven Traffics einzuordnen, weil in dem Moment keine Nutzer auf die eingehenden Emails des Mailinglistenservers warten. Daher kann das Abarbeiten der Mailingliste verlangsamt erfolgen, sofern Bandbreite für andere wichtigere Datenübertragungen benötigt wird.

2.2.3 Internetnutzer

Wie bereits in der Einleitung erwähnt, verfügen mittlerweile immer mehr normale Internetnutzer über gute Internetzugänge mit einer hohen Bandbreite und immer schnelleren Downloads.

Bestes Beispiel hierfür ist T-DSL. Die Telekom bietet einen Zugang ins Internet mit einem Downstream von 768 kBit und einem Upstream von 128 kBit. Allein der Downstream dieser Verbindung entspricht ca. ein Drittel der Bandbreite der 2,3 MBit Leitung. Desweiteren bietet die Telekom auch DSL mit einer Dowloadbandbreite von bis zu 1500 kBit an.

Theoretisch können also bereits drei normale T-DSL-Nutzer bzw. zwei T-DSL-1500-Nutzer, sofern sie mit voller Bandbreite bei der JPBerlin runterladen, die komplette Bandbreite der S-DSL-Leitung belegen.

2.3 Technische Beschränkungen

Das Maximum an Übertragungsbandbreite der JPBerlin wird nicht nur von der S-DSL-Leitung auf 2,3 MBit limitiert. Dabei spielen auch andere Faktoren eine Rolle, Hauptfaktor hierbei ist der Standort der Firma.. Der Carrier der Leitung, KKF-Net, kann aus physikalischen Gründen in der Region maximal 2,3 MBit anbieten. Die JPBerlin kann keine höhere Bandbreite einkaufen, um eine vollausgelastete Leitung auszugleichen. Der einzige Weg ist also die optimale Nutzung der vorhandenen Ressourcen.

3 Soll-Analyse

Die genannten Probleme der beiden Firmen sollen in zwei Schritten angegangen bzw. behandelt werden. Im ersten Schritt soll ein Bandbreitenmanagement für die Aufteilung der Bandbreite der Leitung auf beide Firmen eingeführt und im zweiten Schritt soll der Datenverkehr durch Traffic-Shaping ausgewertet, behandelt und optimiert werden.

3.1 Bandbreitenmanagement

Jedem der beiden Internet Service Provider muss Bandbreite entsprechend der anteiligen Bezahlung der Leitung garantiert werden, um seine Internet-Dienste anbieten zu können, selbst wenn der jeweils andere ISP gerade ein hohes Traffic-Aufkommen hat.

Der Bezahlung entsprechend muss die Aufteilung der Bandbreite ein Drittel für BerliNet und zwei Drittel für die JPBerlin sein.

Ein Internet Service Provider nutzt nicht immer seine volle Bandbreite aus. Wenn einer der beiden ISP's seine Bandbreite nicht ausnutzt, so muss, trotz der 1/3 zu 2/3 Aufteilung, die Möglichkeit gegeben sein, dass der andere ISP diese freie Bandbreite nutzen kann, sofern dieser mehr Bandbreite benötigt.

Die Qualitätssicherung muss also ergeben, dass je nach Auslastung der Leitung jeder Provider die volle 2,3 MBit nutzen kann, mindestens aber seinen entsprechenden ein Drittel / zwei Drittel Anteil hat, selbst unter Volllast.

3.2. Differenzierung und Behandlung von Traffic

Selbst wenn jetzt jedem der beiden Firmen eine garantierte Bandbreite zur Verfügung steht, können auf jedem ISP-Strang immer noch große unwichtige Datentransfers, Anfragen mit hoher Priorität, wie z.B. Webanfragen und interaktive Sitzungen, verlangsamen. Deshalb ist es wichtig Daten nach ihrer Priorität einzustufen und zu transportieren.

3.2.1 Differenzierbarkeit nach Ports

Traffic lässt sich z.B. bereits durch die Port-Nummern unterscheiden, zum wichtigen Traffic gehören SSH-Sitzungen (Shell), die den Port 22 nutzen. Ebenfalls dazu sollten Webanfragen über Port 80 gehören. Weiterhin werden so möglichst alle Ports als wichtig eingestuft, die durch direkte Verbindungen von normalen Internetuser genutzt werden.

Entsprechend wird weniger wichtiger Datenverkehr mit geringer Priorität behandelt. Dazu zählen Verbindungen der Server zu anderen Servern, bei denen unter Umständen große Datenmengen auf einmal versendet bzw. empfangen werden, oder Verbindungen, die Nutzer im Alltag selten nutzen (FTP-Upload von Webseiten). Im Beispiel eines Mailinglistenservers kann beim Versenden der Port 25 als minder prioritär klassifiziert werden.

Genauso ist es mit den News-Servern der JPBerlin (NNTP). Die News-Server tauschen ständig über den Port 119 die neusten Daten mit anderen News-Servern aus, was als permanente Grundbelastung auftritt. Das Austauschen der neusten News kann dabei aber auch wie Emails mit Verlangsamung erfolgen, d.h. die Übertragungsgeschwindigkeit ist dabei egal.

Bei Volllast des Uplinks müssen Verbindungen der wichtigen Ports signifikant mehr Datendurchsatz haben, als andere Dienste.

3.2.2 Differenzierbarkeit nach IP-Adressen

Eine weitere Möglichkeit ist Datenverkehr über die Quell- und Ziel-IP-Adressen auszuwerten. Darüber lässt sich festlegen, von welchen Computern mit welcher Wichtigkeit die zu transportierenden Daten behandelt werden. Hierbei lassen sich Server bereits durch ihre IP-Adresse als wichtig oder minder wichtig einstufen.

Im Beispiel Mailinglistenserver „ilpostino.jpberlin.de“ mit der IP-Adresse 62.8.206.157 lässt sich über die Source-IP festlegen, dass er von vornherein weniger wichtig ist und das Versenden und Empfangen der Emails bzw. Abarbeiten der Mailingliste mit Verzögerung erfolgen kann.

Die Qualitätsüberprüfung muss analog zur Portdifferenzierung Verbindungen mit wichtigen IP-Adressen mehr Datendurchsatz bereit gestellt werden.

3.2.3 Differenzierbarkeit durch Type of Service

Außerdem lässt sich Datenverkehr noch durch ein ToS-Flag („Type of Service“) differenzieren. Dieses Flag wird von bestimmten Applikationen automatisch gesetzt und steht in dem Header eines IP-Frames. Dabei wird ein hexadezimaler Zahlenwert in den IP-Header geschrieben, der aussagt, mit welchen Anforderungen transportiert werden soll. Bei einer SSH-Sitzung wird beispielsweise das Datenpaket, durch SSH selbst, mit 0x10 als ToS-Flag markiert, was besagt, dass es sich um interaktiven Verkehr handelt und mit einer möglichst geringen Verzögerung übertragen werden soll – sofern möglich.

Die folgende Tabelle stellt die Bitkombinationen von „Typ of Service“-Flags dar:

ToS-Flag	Bedeutung
0x0	Normaler Dienst
0x2	Minimale Kosten
0x4	Maximale Zuverlässigkeit
0x6	Minimale Kosten + Maximale Zuverlässigkeit
0x8	Maximaler Durchsatz
0xa	Minimale Kosten + Maximaler Durchsatz
0xc	Maximale Zuverlässigkeit + Maximaler Durchsatz
0xe	Minimale Kosten + Maximaler Durchsatz + Maximale Zuverlässigkeit
0x10	Geringe Verzögerung
0x12	Minimale Kosten + Geringe Verzögerung
0x14	Maximale Zuverlässigkeit + Geringe Verzögerung
0x16	Minimale Kosten + Maximale Zuverlässigkeit + Geringe Verzögerung
0x18	Maximaler Durchsatz + Geringe Verzögerung
0x1a	Minimale Kosten + Maximaler Durchsatz + Geringe Verzögerung
0x1c	Maximale Zuverlässigkeit + Maximaler Durchsatz + Geringe Verzögerung
0x1e	Minimale Kosten + Maximale Zuverlässigkeit + Maximaler Durchsatz + Geringe Verzögerung

Verbindungen vom ToS-Typ „0x10“ , vor allem SSH-Shell-Sitzungen, sollen auch bei hoher Auslastung der Bandbreite flüssig funktionieren, die Auswertung der ToS-Anforderung „maximaler Datendurchsatz“ ist aus Sicht der JPBerlin kein gewünschtes Ziel.

3.2.4 Acknowledge Pakete

Bei jedem Datentransfer werden nicht nur die zu sendenden Daten übertragen, sondern es werden so genannte Acknowledge Pakete (ACK-Pakete) vom Empfänger an den Absender zur Bestätigung geschickt. Das ist eine reine Sicherheitsfunktion, die ja gerade TCP/IP ausmacht, um zu gewährleisten, dass

die Daten vollständig erhalten bzw. versendet wurden. Damit der Sender nicht immer auf den Empfang von ACK-Paketen warten muss, wird eine Anzahl von Datenpaketen festgelegt, die gesendet werden können, ohne dass ein ACK-Paket erhalten werden muss („Window-Size“). Ist der „Vorsprung“ zu groß, also größer als die Window-Size, muss der Sender warten, bis neue ACK-Pakete eintreffen.

Das Problem dabei ist, dass bei einem Download immer ACK-Pakete gesendet werden müssen. Transportiert man nebenbei noch Daten in die Gegenrichtung, werden ACK-Pakete nicht mehr so schnell bzw. zeitig versendet, weil sie sich zwischen den anderen Daten der Gegenrichtung einreihen müssen. Am anderen Ende der Verbindung bleiben ACK-Pakete aus und die Geschwindigkeit des Datentransfers wird unnötig verringert, weil auf Rückbestätigung gewartet werden muss. Der Download wird langsamer.

Dieser Konflikt tritt z.B. dann beim Anwender auf, wenn eine Datei runtergeladen wird und gleichzeitig eine Email versendet werden soll. Die ausgehende Email blockiert ausgehende ACK-Pakete des Downloads, der Download wird langsamer, obwohl die eingehende Leitung eigentlich frei ist.

Es soll möglichst immer Bandbreite vorhanden sein, um Acknowledge-Pakete zu versenden bzw. zu empfangen, damit die Down- bzw. Upload-Geschwindigkeit nicht einbricht, sobald ein weiterer Datentransfer vorhanden ist. Dabei wird keine Bandbreite freigehalten, sondern es soll dynamisch nach ACK-Paketen gefiltert werden, die bevorzugt behandelt und schneller versendet werden.

Weil ACK-Pakete sehr klein sind, geht deren Übertragung eh so schnell, dass es auf andere Transfers keine spürbaren Auswirkungen gibt.

Messungen müssen also ergeben, dass Downloads keinen spürbaren Bandbreiteneinbruch haben, sobald ein Upload parallel erfolgt.

3.3 Realisierungsbedingungen

Um die Umsetzung so kostengünstig wie nur möglich zu machen, soll das Bandbreitenmanagement auf einen normalen Computer erfolgen, der über ein Linux-Betriebsystem verfügt.

Vorteil von Linux ist die Vermeidung von teuren Lizenz-Kosten und dass Linux Open Source ist und nachverfolgbar ist, was auf dem Rechner geschieht. Desweiteren sind recht schnell neue Patches und Sicherheitsupdates verfügbar und frei downloadbar. Außerdem werden durch das „Open-Source-Projekt“ Linux Sicherheitslücken schneller erkannt und behoben, weil kommerzielle Geheimhaltungsinteressen keine Rolle spielen.

Für die beiden Firmen JPBerlin und BerliNet bedeutet das außerdem, dass vorhandene Ressourcen genutzt werden können, da der zentrale Router, der beide Firmen mit der S-DSL-Leitung verbindet, ein auf Linux basierende Firewall ist, auf dem lediglich Bandbreitenmanagement bzw. Traffic Shaping eingerichtet werden muss.

Außerdem sind weder BerliNet, noch die JPBerlin bereit, etwas anderes als Linux zu nutzen, um sich nicht extra einarbeiten zu müssen. Sie meinen, das System perfekt zu kennen. – Auf jeden Fall besser als jede andere Lösung.

4 Technische Lösung

Um eine technische Lösung des geforderten Traffic-Managements zu finden und anschließend zu realisieren, wird zunächst eine Analyse im Hinblick darauf, wie Linux mit TCP/IP zusammenarbeitet, angelegt.

4.1 TCP/IP & Linux

Die gesamte Behandlung von Datenpaketen und der Versand der Daten läuft unter Linux im Kernel ab. Der Kernel besitzt den so genannte TCP/IP-Stack. Der Stack arbeitet wie eine Warteschlange, in den Daten einfach reingelegt werden, aber nach dem FIFO-Prinzip (First In First Out) ausgelesen bzw. versendet werden. Dem FIFO-Prinzip zur Folge werden die Daten, die zuerst reingelegt werden auch als erstes versendet.

Seit dem Linux-Kernel 2.4.x können für die Behandlung der Daten in der Warteschlange bzw. im Stack Regeln festgelegt werden, wie sie abgearbeitet werden sollen. Das macht ein Traffic-Shaping möglich.

Die Festlegung der Regeln für die Behandlung der Daten erfolgt über sog. „qdiscs“, „Filter“ und „Klassen“ und ist wie eine hierarchischen Baumstruktur aufgebaut.

In dieser Baumstruktur wird jede Warteschlange „**qdisc**“ genannt. Dabei nimmt der Stack die Root-Position in der Baumstruktur ein und wird somit als „**root qdisc**“ bezeichnet.

Flow-IDs sind Bezeichner für qdiscs und Klassen. Sie bestehen aus einem numerischen Wert (z.B. 4:) mit einem Gruppenbezeichner als Anhang (z.B. :3 → 4:3). Über **Filter** werden Pakete einer qdisc, anhand der Flow-ID, zugeordnet, wo sie gepuffert werden. Über **Klassen** werden später mit entsprechender Prioritätensetzung die Pakete aus den qdiscs wieder ausgelesen und versand.

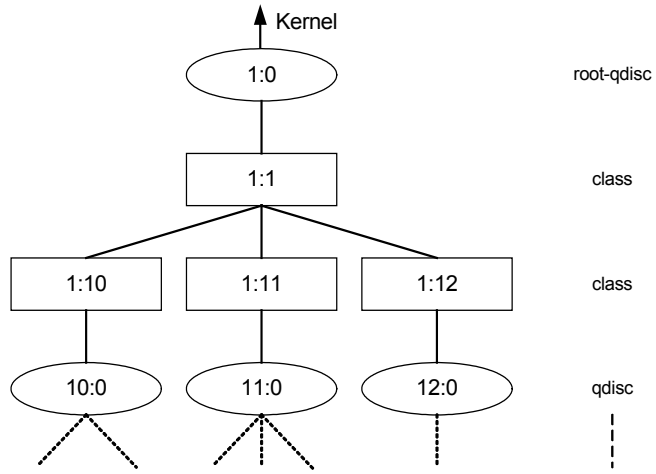

Abb.: Beispiel einer hierarchischen Baumstruktur

4.2 Qdiscs

Qdiscs, oder auch „queue discs“ genannt, ist die Kurzform von queue discipline. Innerhalb der hierarchischen Baumstruktur erhalten qdiscs immer einen ganzzahligen numerischen Bezeichner (Flow-ID) mit dem Anhang Null, wobei der Anhang Null auch weggelassen werden kann (z.B. 4:0 oder 4:).

Bei qdiscs handelt es sich um Warteschlangen, in denen Daten zwischengepuffert werden, bevor sie weiter ausgewertet oder versendet werden. Hinter den qdiscs steckt ein Algorithmus der festlegt, in welcher Reihenfolge sie abgearbeitet werden. Sofern die Regeln nicht verändert wurden, werden die Daten nach dem FIFO-Prinzip abgearbeitet („pfifo_fast“).

Es gibt insgesamt drei Hauptgruppen von qdiscs. Eine der drei wurde bereits erwähnt, die „FIFO-qdisc“. Außerdem gibt es noch die „SFQ-qdisc“ und die „CBQ-qdisc“.

SFQ steht für „Stochastic Fairness Queueing“. Ziel dieses Verfahrens ist eine faire Behandlung der Daten für alle Verbindungen. SFQ arbeitet nach einem verbindungsorientierten Round-Robin Verfahren, bei dem jede Verbindung mit einer gehashten Verbindungs-ID markiert und nach gesondert betrachtet wird.

Der entscheidende Unterschied der SFQ-qdiscs zu den „FIFO-qdiscs“ besteht darin, dass es bei einer „FIFO-qdiscs“ Verbindungen mit vielen Paketen den Verbindungen mit wenigen Paketen die Bandbreite rauben, also die qdiscs „fluten“, so dass ungerecht Bandbreite verteilt wird. Bei SFQ-qdiscs jedoch werden für jede Verbindung jeweils eigene qdiscs angelegt, die gleichberechtigt abgearbeitet werden, ganz egal wie viele Pakete eine Verbindung nun tatsächlich produziert und in die qdiscs einspeist.

SFQ-qdiscs werden meistens am Ende einer hierarchischen Struktur benutzt, da erst hier die Pakete tatsächlich gepuffert werden und der Verteilungskampf um die Bandbreite unter den einzelnen Datenpakete und Verbindungen zu Tragen kommt.

Beispiel:

```
# BerliNet
tc qdisc add dev eth0 parent 1:3 handle 30: sfq perturb 10
# JPBerlin
tc qdisc add dev eth0 parent 1:4 handle 40: sfq perturb 10
```

In diesem Beispiel wurden zwei SFQ-qdiscs angelegt, eine für BerliNet und eine für die JPBerlin. Diese beiden qdiscs haben als hierarchische „Eltern“ (parent) die Klasse „1:3“ und „1:4“ und hängen von der Schnittstelle eth0 ab. Die qdiscs selber haben die Flow-ID „30:“ und „40:“. Der Hashwert der Verbindungs-ID wird dabei alle 10 Sekunden erneuert („perturb 10“) – für uns nicht weiter wichtig.

CBQ ist die Abkürzung für „Class Based Queueing“. Dieses Verfahren arbeitet nach dem Prinzip, dass Traffic über Klassen markiert wird und dementsprechend eingestuft und unterschiedlich behandelt wird. CBQ-qdiscs folgen meistens direkt auf Klassen und werden oft auch direkt während der Definition von Klassen mit angelegt.

Über Klassen lässt sich die Behandlung bzw. Abarbeitungsweise von Datenpaketen festlegen, wenn sie aus den cbq-qdiscs ausgelesen und versendet werden.

Eine Ausnahme ist die root-qdisc. Sie folgt nicht auf eine Klasse und wird „selbständig“ angelegt.

Beispiel:

```
tc qdisc add dev eth0 root handle 1: cbq avpkt 1000 bandwidth
10mbit cell 8
```

In dem Beispiel wird die root-qdisc nach CBQ angelegt. Sie liegt direkt an der Schnittstelle eth0 als root und erhält die Flow-ID „1:0“, alias „1:“ und hat eine Bandbreite von 10 MBit bei einer durchschnittlichen Paketgröße von 1000 Bytes („avpkt 1000“).

4.3 Filter

Filter erkennen und ordnen eintreffende Datenpakete in die unterschiedlichen qdiscs ein, wo sie dann gepuffert werden, bevor sie versendet werden. Dabei kann z.B. nach IP-Adressen, Type-of-Service-Flag (ToS) und nach Port-Nummern gefiltert werden.

Beispiel:

Im folgenden Beispiel wird ein Filter für Source-IP-Adressen aus dem Subnetz 62.8.206.128/27 auf die Flow-ID "1:12" angelegt.

```
# JPBerlin-Netz
tc filter add dev eth0 parent 1: protocol ip prio 2 u32 \
         match ip src 62.8.206.128/27 flowid 1:12
```

In dem nächsten Beispiel wird ein Filter angelegt, der bei IP-Paketen im IP-Header das Type-of-Service-Flag auswertet. Die Flow-ID wird dabei auf „30:1" gesetzt. Hier wird auf das ToS-Flag mit dem Wert 0x10 überprüft

```
# TOS Minimum Delay (ssh, NOT scp) in 1:12
tc filter add dev eth0 parent 1:1 protocol ip prio 10 u32 \
      match ip tos 0x10 0xff  flowid 1:12
```

4.4 Klassen

Klassen sind für den ausgehenden Traffic zuständig. Damit lässt sich die Behandlung von Daten regeln, die aus qdiscs ausgelesen und versendet werden. Über Klassen lässt sich auch die Bandbreite zum Versenden und die Reihenfolge festlegen, was zuerst gesendet werden soll.

Beispiel:

```
UPLINK=2000
# Klasse 1:3 ist BerliNet mit 1/3 Bandbreite
# Klasse 1:4 ist JPBerlin mit 2/3 Bandbreite
tc class add dev eth0 parent 1:1 classid 1:3 cbq rate
           $[1*$UPLINK/3]kbit allot 1600 prio 1 avpkt 1000
tc class add dev eth0 parent 1:1 classid 1:4 cbq rate
           $[2*$UPLINK/3]kbit allot 1600 prio 1 avpkt 1000
```

In diesem Beispiel werden zwei Klassen an der Schnittstelle eth0 mit dem Parent „1:1" angelegt. Die Klassen erhalten die Class-IDs / Flow-IDs „1:3" und „1:4", für

die Klasse „1:3“ wird ein Drittel der Bandbreite bereitgestellt und für die Klasse „1:4“ wird zwei Drittel zur Verfügung gestellt.

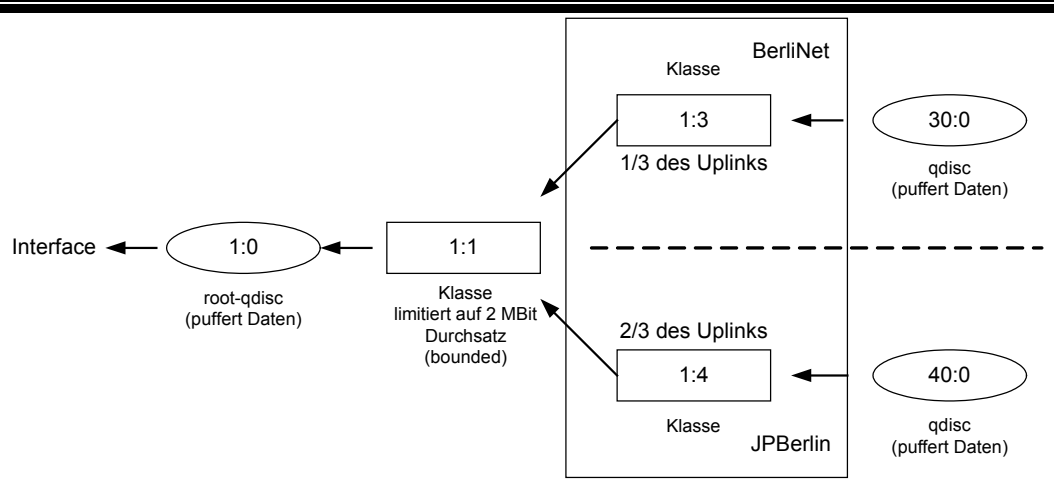

Abb.: Beispiel für Bandbreitenmanagement

4.5 Das „Zoll-Beispiel“

Die Arbeitsweise von Filtern, Klassen und qdiscs entspricht beispielsweise der Zollabfertigung an der Grenze. Der Verkehr kommt über die Autobahn in beliebiger Reihenfolge an die Grenze heran und soll durch den Zoll. Dabei ist bisher keine Unterscheidung zwischen LKWs und PKWs gemacht worden (siehe Abb1). Jedoch dauert die Abfertigung der LKWs länger als die der PKWs und dem entsprechend müssen die PKWs unnötig lange warten, bis die vor ihnen stehenden LKWs abgefertigt wurden.

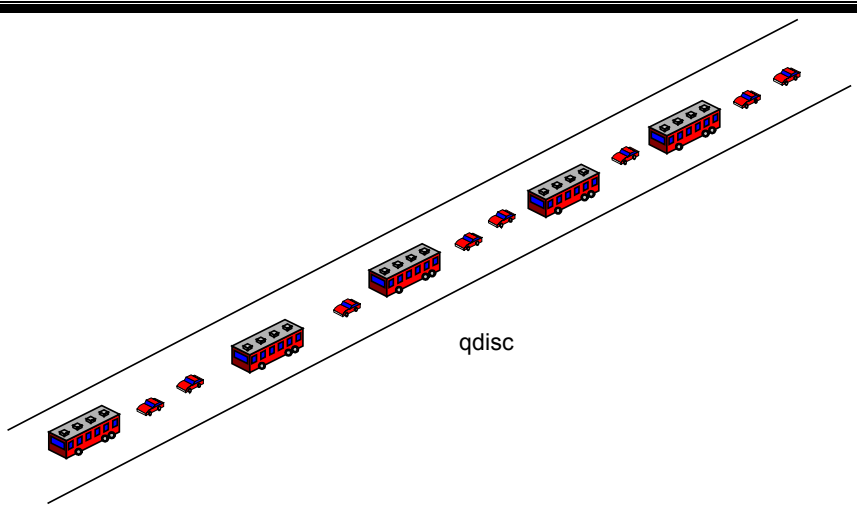

Abb1.: Beispiel für nur eine Warteschlange

Sind zwei getrennte Warteschlangen vorhanden, können Abarbeitungsregeln („Klassen“) angewendet werden. Dann können während ein langwieriger LKW bearbeitet wird durchaus schnell drei PKW’s abgearbeitet werden und somit ist das „Durchschleusen“ der PKWs bei weitem nicht so zeitaufwendig und LKWs haben keine wirkliche Zeitverzögerung (siehe Abb2).

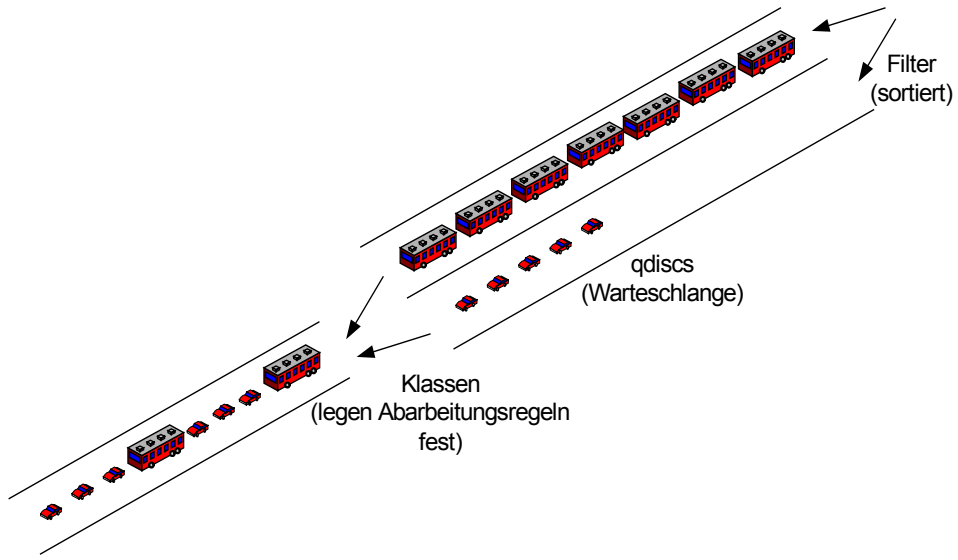

Abb2: Beispiel für getrennte Warteschlangen mit Abarbeitungsregel

5 Technische Umsetzung

Zur Realisierung einer technischen Umsetzung suchte ich zuerst nach einer Ausgangsbasis und überprüfe sie auf Tauglichkeit für die Zwecke eines Traffic-Managements, also Bandbreitenaufteilung und Traffic-Shaping, wie die beiden Internet Service Provider es benötigen.

5.1 Vorhandenes und Recherche

Durch Recherchen über das Internet habe ich ein Skript gefunden, das sog. „Wondershaper-Skript“. Es ist ein weitestgehend fertig konfiguriertes Skript, das nur durch geringfügiges Anpassen einsatzfähig ist. Dieses Skript arbeitet mit den zuvor genannten Befehlen von „tc“.

Problematisch am „Wondershaping-Skript“ ist, dass es den „zweistufigen“ Anforderungen, die die beiden Internet Service Provider und Webhoster stellen, nicht entsprechen. Es führt lediglich ein normales Traffic-Shaping ohne Bandbreitenmanagement aus und ist somit leider nur „einstufig“.

Desweiteren ist es darauf ausgerichtet ein Traffic-Shaping für normale Internetnutzer, speziell für DSL-Nutzer, zu realisieren, um die wenigen Ressourcen zu optimieren.

Daher ist es für unsere Zwecke nahezu unbrauchbar. Es muss komplett umgeschrieben werden, um Traffic-Shaping für zwei ISP-Stränge mit davor geschaltetem Bandbreitenmanagement zu koordinieren. Es dient aber als gutes Vorbild und als Ausgangsbasis.

5.2 Planung der hierarchische Baumstruktur

Um die „zweistufigen“ Anforderungen der beiden Internet Service Providern in einem Skript, basierend auf „tc“-Befehlen, zu entwickeln, wird zuvor eine hierarchische Baumstruktur erstellt, an der man sich beim Schreiben des Traffic-Management-Skriptes orientieren kann.

Die folgende Grafik stellt die für den Lösungsweg erstellte Baumstruktur nach dem Prioritätsprinzip von „tc“ dar.

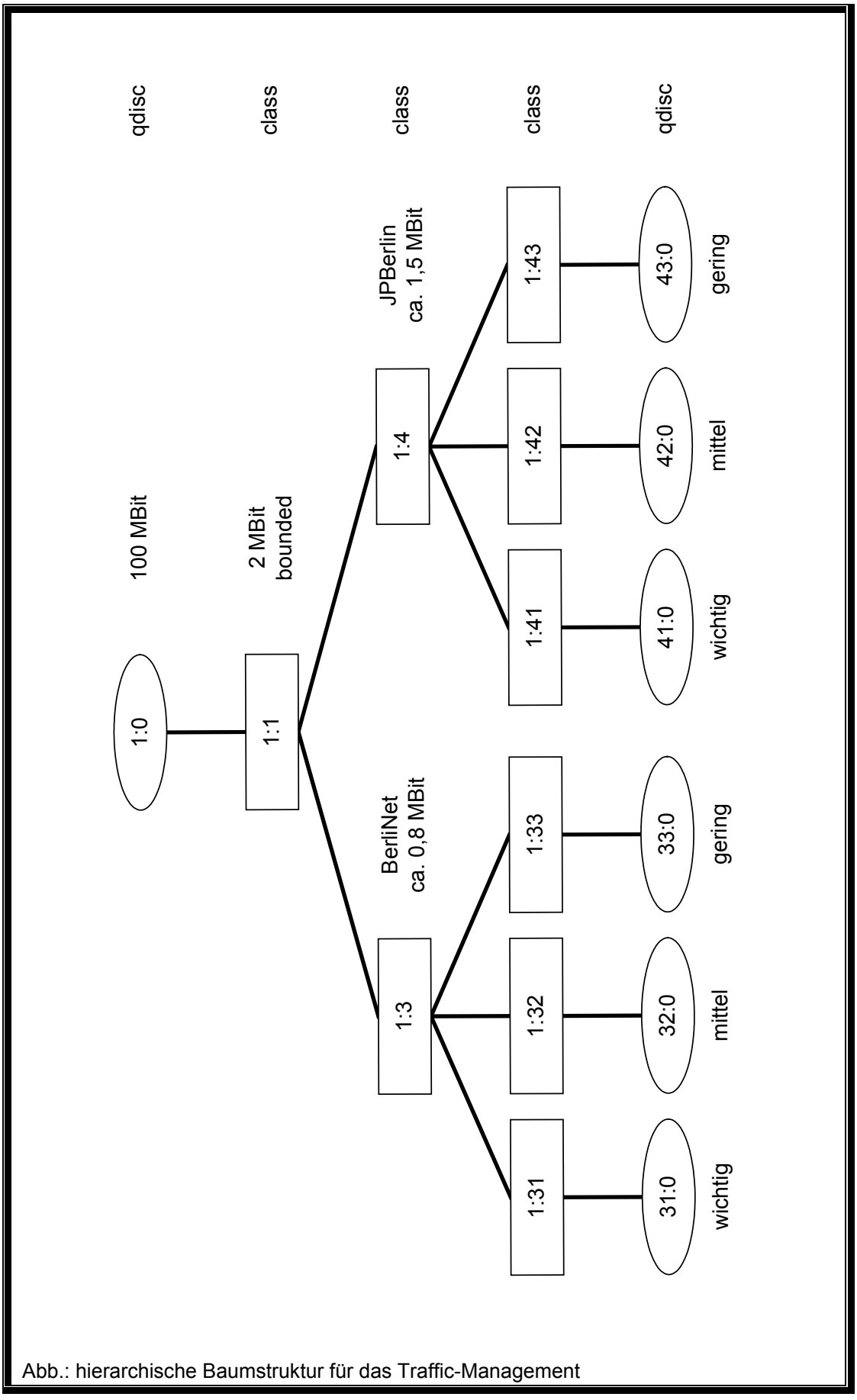

Abb.: hierarchische Baumstruktur für das Traffic-Management

5.3 Implementierung in Skript-Form

Das in Anhang A abgedruckte Skript bildet diese Baumstruktur in tc-Befehlen nach in dem es die benötigte root-qdisc, sowie alle darunter liegenden qdiscs und Klassen einrichtet.

Danach werden die benötigten Filter eingerichtet, die die Datenpakete entsprechend der oben formulierten Soll-Ziele (siehe 3.2.1 bis 3.2.4) in die qdiscs einsortiert.

Es muss dafür gesorgt werden, dass das Skript auf dem Gateway bei jedem Systemstart automatisch geladen wird, um das Shaping einzurichten, da die mit tc vorgenommenen Regeln einen Reset oder Neustart des Linux-Systems nicht überleben (z.B. Startskript in /etc/init.d/).

Im laufenden Betrieb kann das Skript jederzeit manuell an- bzw. abgeschaltet werden. Das Traffic-Management ist ein sog. Bash-Skript; welches direkt über „traffic-mangement [Option]“ aufrufbar ist. Sofern sich das Skript in dem Verzeichnis „/usr/local/bin/“ befindet, kann es wie jeder andere Linux-Befehl direkt aufgerufen werden, man muss sich also nicht im gleichen Verzeichnis befinden, wie das Skript selber.

Durch die Option „status“, die einfach als Argument übergeben wird, wird eine Liste mit allen qdiscs, Filtern und Klassen mit ihrem Status angezeigt. Hier kann man u.a. auch ablesen, wie viele Datenpakete durch welche Klasse, qdisc oder Filter „gegangen“ ist und wie viel an welcher Stelle „geliehen“ worden ist.

Die Option „stop“, die ebenfalls als Argument an das bash-Programm übergeben wird, löscht alle Einträge der Kernelfunktion „tc“, also Klassen, Filter und qdiscs, und beendet somit das Traffic-Management.

5.4 Messtechnik

Um die technische Implementierung auf dem Haupt-Router der beiden ISPs durchführen zu können, muss zuvor die richtige Funktionsweise des Skriptes gewährleistet werden, da ansonsten die Einführung eines nicht richtig funktionierenden Skriptes Schaden für die beiden Provider bedeuten könnte. Zur Bestimmung und Bestätigung der Funktionalität des „zweistufigen“ Traffic-Managements müssen Messungen der einzelnen Filter und Klassen gemacht werden. In den Messungen stellt sich nicht nur die richtige Funktionsweise und umwerfende Auswirkung des Skriptes heraus, sondern verdeutlicht zunehmend das Verständnis der hierarchischen Baumstruktur und des Syntax des tc-Skriptes.

Um sinnvolle Messungen zu machen, braucht man ein passendes Labor-Modell, mit dem entsprechend den Filter- und Klassen-Regeln verschiedene Szenarien durchgemessen werden, bei denen die genutzte Bandbreite mehrerer Datenübertragungen festgehalten und grafisch ausgewertet wird.

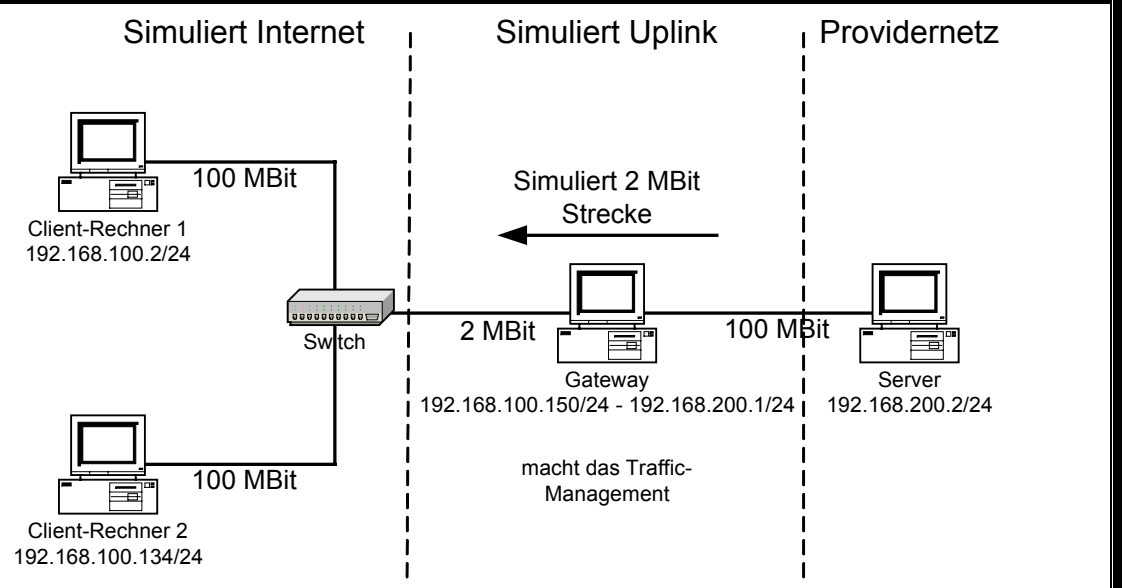

Abb.: Netzwerk-Topologie des Labor-Modells

Durch das Labor-Modell werden Server-Client-Verbindungen, über ein Gateway mit Traffic-Shaping und Messeinrichtung simuliert, mit Hilfe derer Verbindungen, wie unter realen Bedingungen nachgestellt und somit gemessen werden können. Um die realen Bedingungen der S-DSL-Leitung nachzubilden, wurde die Bandbreite auf 2 MBit limitiert.

Auf dem Gateway (Intel Celeron 566 mit 128 MB RAM) wird für jedes Szenario das Traffic-Management-Skript entsprechend angepasst und mit den Client-Rechnern Up- bzw. Downloads durchgeführt, deren Bandbreiten mit Hilfe eines selbstgeschriebenen Skriptes in Ein-Sekunden-Takt mitgeloggt und anschließend grafisch in MS Excel ausgewertet wird.

Das Skript, welches zur Messung dient, funktioniert folgendermaßen: Durch die Kernel-Funktion iptables werden Datenpakete nach IP-Adressen und Ports ausgewertet und markiert. Anschließend wird die versendete bzw. empfangene Datenmenge dieser markierten Pakete im Sekunden-Takt in einer Log-Datei untereinander erfasst. Diese Listen der steigenden Datenmengen wird dann in eine Excel-Tabelle importiert und in die augenblickliche Bandbreite umgerechnet und in einem Graphen dargestellt, um Graphen zu glätten, wird über 5 Sekunden gemittelt.

Natürlich muss das „Daten-Erfassungs-Skript“ extra an jedes Mess-Szenario angepasst werden.

Das Mess-Skript besteht aus drei kleineren Skripten, einem Aufruf-Skript und zwei auszuführende Skripte. Das aufrufende Skript wird in dem Verzeichnis /usr/local/bin/ abgelegt, damit es wie ein normaler Linux-Befehl gestartet werden kann.

Quelltext des „Aufruf-Skriptes“:

```
# Mess-Skript – Aufruf:
# MESSUNG <Option> [Name der Messung]
# Beispiele:      MESSUNG messen
#                 MESSUNG data “Name der Messung“

NAME="$2"

case "$1" in
   messen)
   echo "Initialisierung der iptables"
      bash /root/IPTAB-Counter
   echo "Lösche alte Data-Files"
      rm /tmp/DATA-*
   echo "Starte Messung..."
      bash /root/MESS
   ;;
   data)
      grep ACCEPT /tmp/ftp-up | cut -b 9-19 >> /tmp/DATA-up
      grep ACCEPT /tmp/ftp-down | cut -b 9-19 >> /tmp/DATA-down
      mkdir "/tmp/MESS/$NAME"
      cp /tmp/DATA-* "/tmp/MESS/$NAME/"
   ;;
esac
```

In diesem Bash-Skript wird über Optionen die Messung gestartet, oder eine zuvor gemachte Messung in ein Verzeichnis in zwei unterschiedliche Dateien geschrieben.

Die Option „messen“ startet ein weiteres Skript „IPTAB-Counter“, welches „iptables“ initialisiert, damit bestimmte Datenpakete für die Messung markiert werden. Anschließend werden eventuelle vorangegangene Messdaten gelöscht und das „MESS“-Skript wird ausgeführt, welches die eigentliche Erfassung der Daten macht.

Die Option „data“ filtert die zuvor gemachte Messung und schreibt die Ergebnisse in ein durch ein Argument angegebenes Verzeichnis in der Verzeichnisstruktur „/tmp/MESS/“ in zwei Dateien.

Das Initialisierungsskript „IPTAB-Counter“ richtet iptables für jede Messreihe unterschiedlich ein, um Datenpakete entsprechend für jede Messung markieren zu lassen. Da das Skript immer verändert werden muss, wird die jeweils veränderte Passage des Quelltexts immer im Bereich jeder Messung zusätzlich angegeben. Allgemein löscht es die vorhanden iptables-Regeln und legt neue Regel-Ketten an, die spezifisch für jede Messung passende Pakete markiert.

Quelltext „IPTAB-Counter“:

```
# IPTAB-Counter
# aendert die Regeln von iptables, um bestimmte Datenpakete
# fuer die Auswertung zu markieren
```

```
#loescht vorhandene Regeln
iptables -F
iptables -X
echo 1

# legt zwei neue Regelketten an
iptables -N ftp-up
iptables -N ftp-down
echo 2

# Schaltet die Firewall auf Durchzug
iptables -P FORWARD ACCEPT
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
echo 3

# Hier die Regeln fuer die zumarkierenden Datenpakete einfuegen
# -- Messregeln Anfang
# iptables -A FORWARD -s <IP> -p TCP --dport <Port> -j ftp-up
# iptables -A FORWARD -d <IP> -p TCP –sport <Port> -j ftp-down
# -- Messregeln Ende

# Filtert und zaehlt alle Pakete dieser Regel
iptables -A ftp-up -p TCP -j ACCEPT
iptables -A ftp-down -p TCP -j ACCEPT
```

Das „MESS“-Skript macht ein Reset auf alle bereits geloggten Daten durch iptables und schreibt die transferierte Datengröße aller durch die angelegten Regel-Ketten markierten Pakete in zwei Dateien. Diese Erfassung der Daten erfolgt in einer Endlosschleife im Sekundentakt und wird nur durch Abbruch des Prozesses durch „Strg+C“ beendet.

Quelltext des „MESS“-Skriptes:

```
# MESS-Skript
# Erfasst die transferierte Datenmenge der markierten Datenpakete

iptables -Z
rm /tmp/ftp-*

while [ 5 = 5 ] ; do
      iptables -nvxL ftp-up >> /tmp/ftp-up
      iptables -nvxL ftp-down >> /tmp/ftp-down
      sleep 1s
done
```

5.5 Die Messungen

Um die richtige Funktionalität des Traffic-Managements zu bestätigen, müssen auf die Schwerpunkte des Managements angepasste Szenarien, „Kleine Pakete und ACK“, „Ports/Hosts“ und „Type of Service“, durchgemessen und ausgewertet

werden. Das Erreichen der oben definierten Soll-Ziele (3.2.1 bis 3.2.4) muss in entsprechenden Messungen überprüft und nachgewiesen werden.

5.5.1 Messung „Kleine Pakete und ACK“ (3.2.4)

In dieser Messreihe werden zwei Messung zur Erkennung der Wirkung des Traffic-Managements durchgeführt. Die Messungen werden einmal ohne Traffic-Shaping und einmal mit aktiven Shaping durchgeführt, um die Unterschiede vom normalen Bandbreitenverhalten und geshapten Verhalten bewerten zu können.

Um die Daten genau erfassen zu können, werden die iptables-Regeln im „IPTAB-Counter“-Skript in dem auskommentierten Bereich um folgende Zeilen ergänzt:

```
iptables -A FORWARD -s 192.168.100.2 -p TCP --dport 20 -j ftp-up
iptables -A FORWARD -d 192.168.100.134 -p TCP --sport 20 -j ftp-down
```

Die Messungen werden anhand zweier Datenströmen erfasst, einem FTP-Download und einem FTP-Upload.

In dieser Erweiterung der iptables-Regeln, werden die eingehenden Daten von 192.168.100.2 auf Port 20 mit „ftp-up“ und die ausgehenden Daten zur 192.168.100.134 von Port 20 mit „ftp-down“ markiert.

Um das normale Bandbreitenverhalten zu sehen, wird zuerst die Messung ohne Traffic-Shaping durchgeführt.

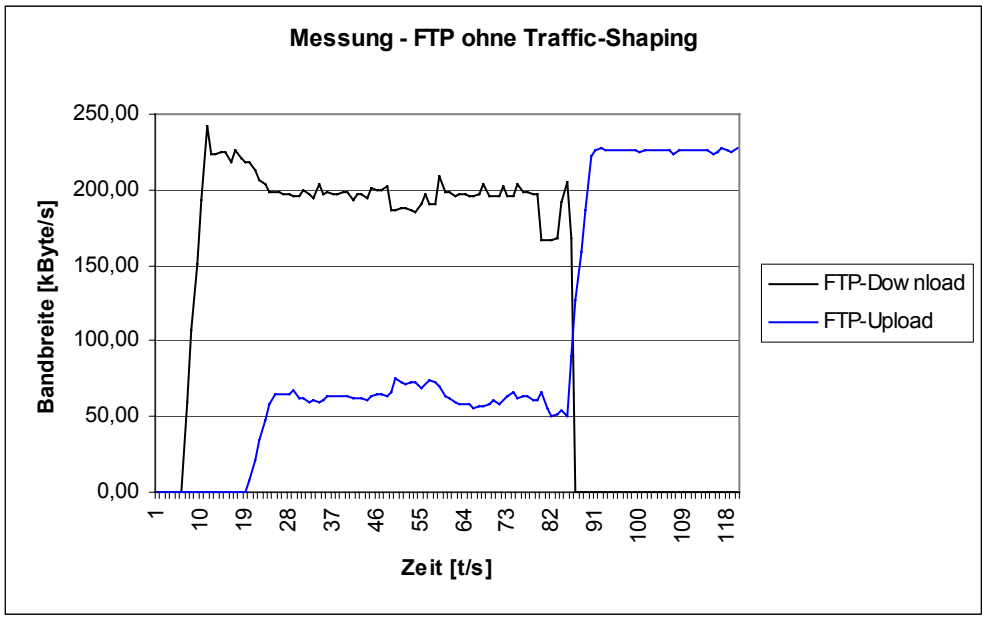

Abb.: Grafische Auswertung der Messung: FTP ohne Traffic-Shaping

Entscheidende Erkenntnis dieser Messung ist der sichtbare Einbruch der Downloadbandbreite zu dem Zeitpunkt, als der Upload einsetzte. Grund dafür ist, dass sich die Acknowledge-Pakete des Downloads zwischen die Upload-Pakete einreihen müssen und schließlich verspätet ankommen, Effekt ist der langsamere Download. Desweiteren ist der FTP-Upload recht langsam, da sich die ACK-Pakete des Uploads zwischen die Download-Pakete einreihen und somit die gleiche Wirkung zustande kommt.

Anschließend wurde die Messung für dieses Szenario mit eingeschaltetem Traffic-Shaping durchgeführt.

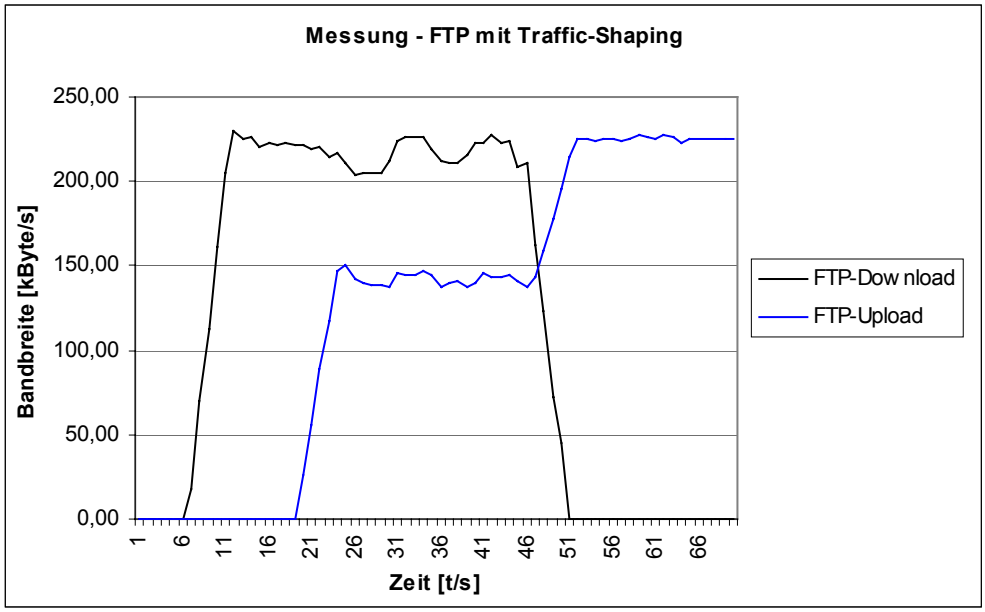

Abb.: Grafische Auswertung der Messung: FTP mit Traffic-Shaping

In dieser Messung mit aktivem Traffic-Shaping sieht man deutlich, dass die Bandbreite des FTP-Downloads beim Einsetzten des FTP-Upload kaum einbricht, außerdem ist die Upload-Bandbreite viel höher, als in der Messung ohne Traffic-Shaping, da hier kleine Pakete, wie Acknowledge-Pakete, bevorzugt behandelt werden und daher nicht verspätet versendet bzw. empfangen werden.

Zu kleinen Paketen gehören auch Daten aus dem Internet Control Message Protocol (ICMP). Um diese Messung zu realisieren, wurde der Server durchgehend von einem Client-Rechner gepingt (ICMP echo request). Um das genaue Verhalten zu beobachten bzw. zu messen wurde zunächst ohne Traffic-Shaping gemessen. Dabei wurde ein Download gestartet und die Pingzeit stieg enorm an (bei ca. 27 Sekunden) . Beim Aktivieren des Traffic-Managements (bei ca. 37 Sekunden) normalisierten sich die Pingzeiten auf ein Niveau, als wäre kein Datentransfer vorhanden. Sobald das Skript wieder abgeschaltet wird, stiegen die Pingzeiten wieder an, bis der Download beendet wird.

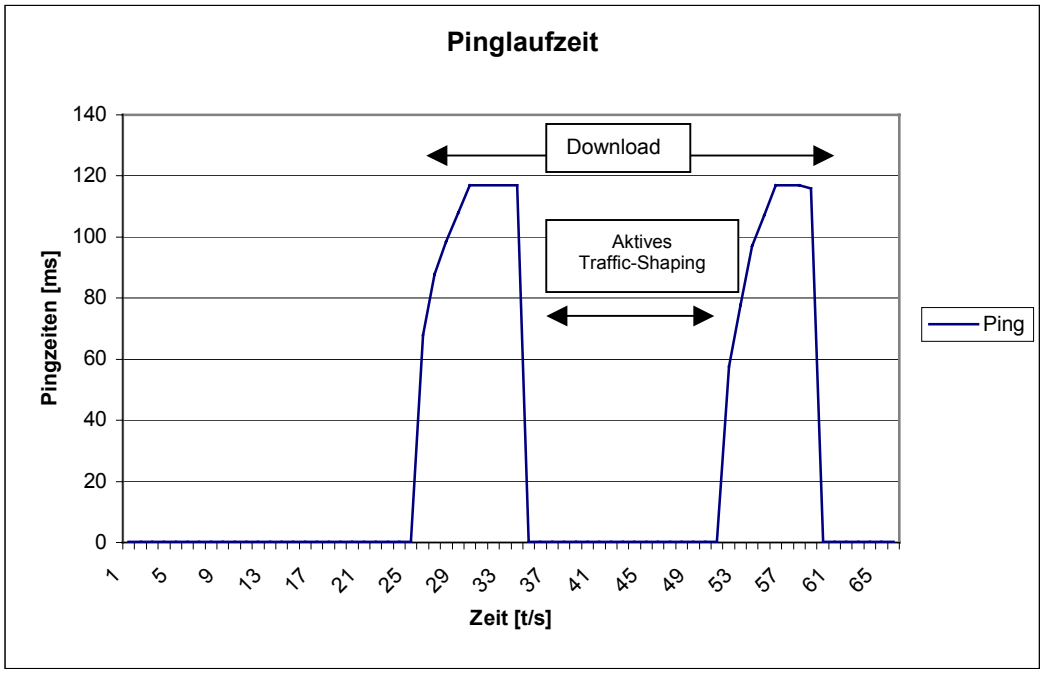

Abb.: Grafische Auswertung der Messung: Pingzeiten und Datentransfer

5.5.2 Messung „Ports / Hosts“ (3.2.1 / 3.2.2)

In der Messung dieses Szenarios stellt sich die Wirkung der Priorisierung von Ports bzw. Hosts des Traffic-Managements heraus. Ebenfalls wird die Messreihe einmal ohne und einmal mit aktivem Traffic-Shaping durchgeführt, um die Unterschiede analysieren zu können.

Für diese Messreihe werden in dem Traffic-Management-Skript die Ports 80 und 81 mit unterschiedlicher Priorität festgelegt, Port 80 („Web“) wird als wichtig und Port 81 als unwichtig eingestuft. Außerdem wird ein Apache-Server auf „listening“ für diese beiden Ports konfiguriert, d.h. dass der Webserver auf beiden Ports anfragen akzeptiert. Es werden zwei Downloads über die zuvor genannten Ports gestartet.

Entsprechend wird das „IPTAB-Counter“-Skript wieder angepasst:

```
iptables -A FORWARD -d 192.168.100.2 -p TCP --sport 80 -j ftp-up
iptables -A FORWARD -d 192.168.100.134 -p TCP --sport 81 -j ftp-down
```

Hier werden die Daten der beiden Client-Rechner zur Unterscheidung über die Source-Ports mit „ftp-up“ und „ftp-down“ markiert, um anschließend gemessen werden zu können.

Zuerst wird die Messung wieder ohne aktivem Traffic-Shaping durchgeführt.

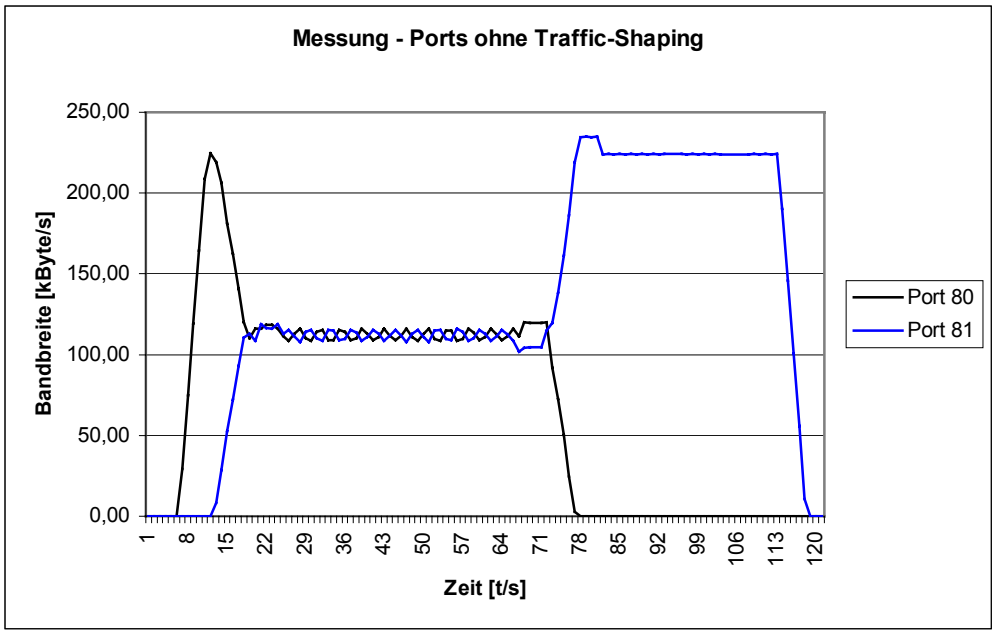

Abb.: Grafische Auswertung der Messung: Ports ohne Traffic-Shaping

In dieser Messung ist gut zu erkennen, dass der erste Download über Port 80 die volle Bandbreite nutzt, bis der zweite Download über Port 81 beginnt. Ab diesen Zeitpunkt teilen sich beide Downloads die Bandbreite zu jeweils 50%, bis der erste Download beendet wird und der zweite die volle Bandbreite nutzt.

Nun als Gegensatz die zweite Messung mit aktivem Traffic-Shaping, um den Unterschied zur Messung ohne Traffic-Shaping beurteilen zu können.

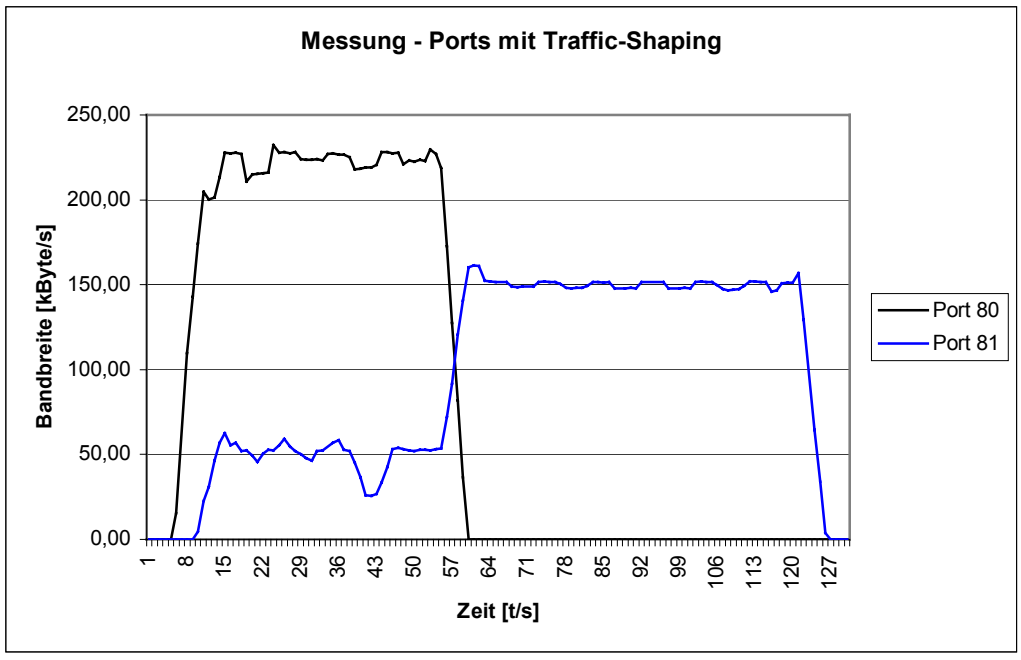

Abb.: Grafische Auswertung der Messung: Ports mit Traffic-Shaping

Diese Messung bei aktiviertem Traffic-Shaping zeigt deutlich, dass die Bandbreite des Downloads über „wichtigen“ Port 80 durchgehend hoch bleibt und die Transportgeschwindigkeit des Downloads über den „unwichtig-deklarierten“ Port 81 so lange niedrig bleibt, bis der wichtige Datentransfer abgeschlossen ist und anschließend enorm ansteigt.

Die Erscheinung, dass der Download auf Port 81 nach beenden des anderen Downloads nur auf eine Bandbreite von ca. 160 kByte/s ansteigt, wurde im Rahmen dieser Messung nicht endgültig geklärt, hängt aber wahrscheinlich mit einem unterschiedlichen Aushandeln der Window-Size zusammen, ist für diese Vergleichsmessung aber nicht weiter relevant.

5.5.3 Messung „Type of Service“ (3.2.3)

Eine tatsächliche Messung zu dem Gesichtspunkt „Type of Service“ zu machen ist sehr aufwendig und daher schwer zu bewerkstelligen. Desweiteren ist in dem verlangten Traffic-Management lediglich die Beschleunigung der mit dem „0x10“-ToS-Flag markierten Datenpakete vorgesehen, speziell geht es um SSH. SSH dient zur Administration der vorhandenen Server und Router des Firmennetzes, ist also eine interaktive Session und soll daher ohne merkbaren zeitlichen Verzögerungen erfolgen.

Die durch das Traffic-Management Beschleunigung der Daten zu erkennen, ist hier lediglich durch eine „subjektive Messung“ möglich, bei der zwei breitbandige Datentransporte parallel zu SSH-Sitzungen erfolgen, um eine eventuelle lange dauernde „Terminalantworten“, also Verzögerungen, zu bemerken.

Meine persönlichen Erfahrungen bei der Durchführung des Tests ist wie folgt beschreibbar: In dem Test ohne Traffic-Shaping waren tatsächlich lange Verzögerungen der SSH-Verbindungen merkbar, mit aktivem Traffic-Shaping hingegen waren keine spürbaren Verzögerungen vorhanden, erfolgten so zu sagen in „Echtzeit“. Das aktive Skript machte die Terminal-Verbindungen verzögerungslos nutzbar und ermöglichte dahingehend ein „unbeschwertes“ Arbeiten, also alle Antworten vom Server erfolgten prompt.

Die Auswirkungen bei SSH ist analog zu der Messreihe mit den Pingzeiten. Bei dieser Messung waren kurze Latenzzeiten erkennbar, die eine SSH-Sitzung erfordert.

6 Fazit

Die Bandbreitenprobleme wurden durch das Traffic-Management spürbar gelöst. Die Leitungsqualität wurde so gesteigert, dass der Erwerb von zusätzlicher Bandbreite auf unbestimmte Zeit in die Zukunft verschoben werden konnte und die JPBerlin entsprechende monatliche Kosten im knapp vierstelligen Bereich sparen kann.

Das Traffic-Shaping beschleunigt Verbindungen zu den Servern auch für normale Internetnutzer spürbar: Webanfragen werden auch zu Stoßzeiten schneller als bisher vom Server beantwortet und die Daten mit höherer Geschwindigkeit ausgeliefert.

Auch externe Benchmarktests verschiedener Firmen bewerten die Geschwindigkeit der Server deutlich höher und ranken die JPBerlin dementsprechend in der Qualität deutlich besser.

Ein schon länger schwelender Konflikt der beiden Provider über die faire Aufteilung der gemeinsam genutzten 2,3 MBit-Leitung wurde durch das Bandbreitenmanagement geregelt.

Die ausgearbeitete Lösung ist durch soweit übertragbar gehalten, dass es durch generelle Anpassungen in anderen Netze eingesetzt werden kann. Auch wenn tiefgehende Kenntnisse des jeweiligen Administrator über die Arbeitsweise von Traffic-Shaping nicht unbedingt zwingend nötig sind, sind grundlegende Verständnisse über Aufbau und Ablauf von Unix-Shell-Skripten vonnöten, um das Script an die eigene Netzstruktur anzupassen.

Es hat sich gezeigt, dass Linux den Aufgaben gewachsen war und als stabile und zuverlässige Lösung für den Einsatz als Traffic-Manager geeignet ist. Der Kernel 2.4 liefert ohne weitere Software alle benötigten Systemfunktionen mit, das Steuer-Programm „tc“ gehört bei allen Distributionen zum mitgelieferten elementaren Minimalsystem.

Das Traffic-Management lässt sich problemlos mit handelsüblicher Computerhardware realisieren. Auch beim Einsatz normaler NoName-Netzwerkkarten zeigte der shapende Intel Celeron 566 keinerlei Performanceprobleme und war für die benötigte Leistung mehr als ausreichend bemessen.

7 abstract

The traffic management has solved the bandwidth problems. The quality of the line has been risen that buying more bandwidth could be pushed into the future for unknown time, so the JPBerlin is able to save monthly costs of an amount of four places.

The traffic shaping accelerates normal internet user's connections to the servers. Webrequests are answered faster than usually, even in "rush-hour", and the data is sent at a higher speed.

Even extern benchmarktests of different companies are telling that the speed of the servers is a lot higher so they rank the quality JPBerlin's servers much better.

The long-taking conflict of the both provider JPBerlin and BerliNet about the fair sharing of the 2.3 mbit line has been solved cause of the included bandwidth management.

The technical solution is transportable that there are only a few changes necessary to use it for other networks. Deeper knowledge of administrator about traffic shaping is not necessary but the usage and knowledge about how shell scripts are working is important to do the changes for the own network.

Developing and testing the traffic management has shown that Linux is a good solution for the use as traffic manager. The kernel 2.4 includes all system requirements; the control program 'tc' is a part of all Linux distributions and belongs to the elementary minimal system.

The traffic management can be easily build up with usual computer hardware. Even the usage of noname network cards showed that the for the shaping used Intel Celeron 566 hasn't had any performance problems and was more than enough as needed cpu power.

Anhang

A Quelltext des Traffic-Management-Skripts

```
#!/bin/bash

#
# Intelligentes Trafficmanagement fuer Internet Service Provider
# Quality of Service mit Linux
# Projektarbeit von Michael Bonin (ITD 4) SS2003
#

# Variablen
DOWNLINK=2000
UPLINK=2000
DEV=eth0

# BerliNet - Filtervariablen:
BLN_NOPRIOHOSTSRC=
BLN_HIGHPRIOHOSTSRC=
BLN_NOPRIOHOSTDST=
BLN_HIGHPRIOHOSTDST=
BLN_NOPRIOPORTSRC=
BLN_HIGHPRIOPORTSRC=
BLN_NOPRIOPORTDST=
BLN_HIGHPRIOPORTDST=

# JPBerlin - Filtervariablen:
JPB_NOPRIOHOSTSRC=
JPB_HIGHPRIOHOSTSRC=
JPB_NOPRIOHOSTDST=
JPB_HIGHPRIOHOSTDST=
JPB_NOPRIOPORTSRC=
JPB_HIGHPRIOPORTSRC=
JPB_NOPRIOPORTDST=
JPB_HIGHPRIOPORTDST=

# End of Var

#
# Nach dieser Zeile nichts veraendern !!!
#

# Auszufuehrende Argumente
if [ "$1" = "status" ]
then
      tc -s qdisc ls dev $DEV
      tc -s class ls dev $DEV
      tc filter show dev $DEV
      exit
fi

# loescht alle zuvor existierenden qdiscs um Fehler zu vermeiden
tc qdisc del dev $DEV root    2> /dev/null > /dev/null
tc qdisc del dev $DEV ingress 2> /dev/null > /dev/null
```

```
if [ "$1" = "stop" ]
then
      exit
fi

# Wenn kein Status oder Stop, dann wird hier weiter gemacht

# installiert root CBQ
# Auf $DEV mit einem 10mbit-Netz mit dem Handle "1:"
tc qdisc add dev $DEV root handle 1: cbq avpkt 1000 bandwidth 100mbit \
      cell 8

# installiert eine CBQ-qdisc auf 2MBit bounded
tc class add dev $DEV parent 1: classid 1:1 cbq bandwidth 100mbit \
       rate ${UPLINK}kbit allot 1500 prio 3 bounded

##########################
# Bandbreitenmanagement: #
# ~~~~~~~~~~~~~~~~~~~~~~ #
##########################

# Aufteilung der Bandbreite von 2 MBit
# BerliNet: Klasse 1:3 - 1/3 der Bandbreite
# JPBerlin: Klasse 1:4 - 2/3 der Bandbreite
tc class add dev $DEV parent 1:1 classid 1:3 cbq rate $[1*$UPLINK/3]kbit \
   allot 1600 prio 1 avpkt 1000
tc class add dev $DEV parent 1:1 classid 1:4 cbq rate $[2*$UPLINK/3]kbit \
   allot 1600 prio 1 avpkt 1000


#Zuweisung der IP-Subnetze auf die Klassen

# BerliNet - Klasse 1:3
tc filter add dev $DEV parent 1: protocol ip prio 2 u32 \
      match ip src 62.8.206.0/25 flowid 1:3
tc filter add dev $DEV parent 1: protocol ip prio 2 u32 \
      match ip src 62.8.206.160/27 flowid 1:3
tc filter add dev $DEV parent 1: protocol ip prio 2 u32 \
      match ip src 62.8.206.192/26 flowid 1:3

# JPBerlin - Klasse 1:4
tc filter add dev $DEV parent 1: protocol ip prio 2 u32 \
         match ip src 62.8.206.128/27 flowid 1:4

# Jeder der beiden hat eine garantierte Bandbreite, aber es kann
# Bandbreite geliehen werden, sofern der jeweils andere seine
# Bandbreite nicht voll ausnutzt.
# D.h. jeder hat den vollen $Uplink sofern der andere keine
# Bandbreite nutzt.


####################
# Traffic-Shaping: #
# ~~~~~~~~~~~~~~~~ #
####################

# Bewirkt eine sinnvollere Ausnutzung der Bandbreite durch
# Auswertung der Daten.

# Dazu richten wir wieder abgeleitete Unterklassen ein, die Anteile von
# den Parentklassen kriegen.
```

```
# BerliNet
# Prioritaet: hoch
tc class add dev $DEV parent 1:3 classid 1:31 cbq rate ${UPLINK}kbit \
   allot 1600 prio 1 avpkt 1000
# Prioritaet: mittel
tc class add dev $DEV parent 1:3 classid 1:32 cbq rate $[9*$UPLINK/10]kbit \
   allot 1600 prio 2 avpkt 1000
# Prioritaet: niedrig
tc class add dev $DEV parent 1:3 classid 1:33 cbq rate $[6*$UPLINK/10]kbit \
   allot 1600 prio 2 avpkt 1000


# JPBerlin
# Prioritaet: hoch
tc class add dev $DEV parent 1:4 classid 1:41 cbq rate ${UPLINK}kbit \
   allot 1600 prio 1 avpkt 1000
# Prioritaet: mittel
tc class add dev $DEV parent 1:4 classid 1:42 cbq rate $[9*$UPLINK/10]kbit \
   allot 1600 prio 2 avpkt 1000
# Prioritaet: niedrig
tc class add dev $DEV parent 1:4 classid 1:43 cbq rate $[6*$UPLINK/10]kbit \
   allot 1600 prio 2 avpkt 1000

# Anlegen der dazugehoerenden Warteschlangen
# Berlinet
tc qdisc add dev $DEV parent 1:31 handle 31: sfq perturb 10
tc qdisc add dev $DEV parent 1:32 handle 32: sfq perturb 10
tc qdisc add dev $DEV parent 1:33 handle 33: sfq perturb 10
# JPBerlin:
tc qdisc add dev $DEV parent 1:41 handle 41: sfq perturb 10
tc qdisc add dev $DEV parent 1:42 handle 42: sfq perturb 10
tc qdisc add dev $DEV parent 1:43 handle 43: sfq perturb 10


############################
# Anlegen der Filterregeln #
############################

############
# BerliNet #
############

# TOS minimale Verzoegerung
tc filter add dev $DEV parent 1:3 protocol ip prio 10 u32 \
      match ip tos 0x10 0xff  flowid 1:31

# Steuerbefehle etc. (z.B. ICMP)
tc filter add dev $DEV parent 1:3 protocol ip prio 11 u32 \
        match ip protocol 1 0xff flowid 1:31

# "beschleunigen" kleiner Pakete
tc filter add dev $DEV parent 1:3 protocol ip prio 12 u32 \
   match ip protocol 6 0xff \
   match u8 0x05 0x0f at 0 \
   match u16 0x0000 0xffc0 at 2 \
   flowid 1:31

# Traffic von unwichtigen IPs
for a in $BLN_NOPRIOHOSTSRC
do
      tc filter add dev $DEV parent 1:3 protocol ip prio 13 u32 \
         match ip src $a flowid 1:33
done
```

```
# Traffic von wichtigen IPs
for a in $BLN_HIGHPRIOHOSTSRC
do
      tc filter add dev $DEV parent 1:3 protocol ip prio 13 u32 \
         match ip src $a flowid 1:31
done

# Traffic an unwichtigen IPs
for a in $BLN_NOPRIOHOSTDST
do
      tc filter add dev $DEV parent 1:3 protocol ip prio 14 u32 \
         match ip dst $a flowid 1:33
done

# Traffic an wichtigen IPs
for a in $BLN_HIGHPRIOHOSTDST
do
      tc filter add dev $DEV parent 1:3 protocol ip prio 14 u32 \
         match ip dst $a flowid 1:31
done

# Traffic von unwichtigen Ports
for a in $BLN_NOPRIOPORTSRC
do
      tc filter add dev $DEV parent 1:3 protocol ip prio 15 u32 \
         match ip sport $a 0xffff flowid 1:33
done

# Traffic von unwichtigen Ports
for a in $BLN_HIGHPRIOPORTSRC
do
      tc filter add dev $DEV parent 1:3 protocol ip prio 15 u32 \
         match ip sport $a 0xffff flowid 1:31
done

# Traffic an unwichtigen Ports
for a in $BLN_NOPRIOPORTDST
do
      tc filter add dev $DEV parent 1:3 protocol ip prio 16 u32 \
         match ip dport $a 0xffff flowid 1:33
done

# Traffic an unwichtigen Ports
for a in $BLN_HIGHPRIOPORTDST
do
      tc filter add dev $DEV parent 1:3 protocol ip prio 16 u32 \
         match ip dport $a 0xffff flowid 1:31
done

# Alles was weder hohe noch geringe Prioritaet hat
tc filter add dev $DEV parent 1:3 protocol ip prio 17 u32 \
   match ip dst 0.0.0.0/0 flowid 1:32


############
# JPBerlin #
############

# TOS minimale Verzoegerung
tc filter add dev $DEV parent 1:4 protocol ip prio 10 u32 \
      match ip tos 0x10 0xff flowid 1:41
```

```
# Steuerbefehle etc. (z.B. ICMP)
tc filter add dev $DEV parent 1:4 protocol ip prio 11 u32 \
        match ip protocol 1 0xff flowid 1:41

# "beschleunigen" kleiner Pakete
tc filter add dev $DEV parent 1:4 protocol ip prio 12 u32 \
   match ip protocol 6 0xff \
   match u8 0x05 0x0f at 0 \
   match u16 0x0000 0xffc0 at 2 \
   flowid 1:41

# Traffic von unwichtigen IPs
for a in $JPB_NOPRIOHOSTSRC
do
        tc filter add dev $DEV parent 1:4 protocol ip prio 13 u32 \
           match ip src $a flowid 1:43
done

# Traffic von wichtigen IPs
for a in $JPB_HIGHPRIOHOSTSRC
do
       tc filter add dev $DEV parent 1:4 protocol ip prio 13 u32 \
          match ip src $a flowid 1:41
done

# Traffic an unwichtigen IPs
for a in $JPB_NOPRIOHOSTDST
do
        tc filter add dev $DEV parent 1:4 protocol ip prio 14 u32 \
           match ip dst $a flowid 1:43
done

# Traffic an wichtigen IPs
for a in $JPB_HIGHPRIOHOSTDST
do
       tc filter add dev $DEV parent 1:4 protocol ip prio 14 u32 \
          match ip dst $a flowid 1:41
done

# Traffic von unwichtigen Ports
for a in $JPB_NOPRIOPORTSRC
do
        tc filter add dev $DEV parent 1:4 protocol ip prio 15 u32 \
           match ip sport $a 0xffff flowid 1:43
done

# Traffic von wichtigen Ports
for a in $JPB_HIGHPRIOPORTSRC
do
       tc filter add dev $DEV parent 1:4 protocol ip prio 15 u32 \
          match ip sport $a 0xffff flowid 1:41
done

# Traffic an unwichtigen Ports
for a in $JPB_NOPRIOPORTDST
do
        tc filter add dev $DEV parent 1:4 protocol ip prio 16 u32 \
           match ip dport $a 0xffff flowid 1:43
done
```

```
# Traffic an wichtigen Ports
for a in $JPB_HIGHPRIOPORTDST
do
       tc filter add dev $DEV parent 1:4 protocol ip prio 16 u32 \
          match ip dport $a 0xffff flowid 1:41
done

# Alles was weder hohe noch geringe Prioritaet hat
tc filter add dev $DEV parent 1:4 protocol ip prio 17 u32 \
   match ip dst 0.0.0.0/0 flowid 1:42
```

B Literaturverzeichnis

„Netzwerkanalyse unter Linux“
Wolfgang Barth / Jens Banning
Grundlagen – Werkzeuge – Techniken
SuSE PRESS
Nürnberg 2002

„Linux Advanced Routing & Traffic Control HOWTO“
Bert Hubert
http://lartc.org/howto/

“Queueing Disciplines for Badwidth Management”
http://lartc.org/howto/lartc.qdiscs.html
Stand: 06. Juni 2003

“Verkehrspolizei – BandbreitenManagement mit Linux”
Jürgen Schmidt
in: c’t, Nr. 24, 2002, Seite 224 ff

„The Wonder Shaper“
Bert Hubert
http://lartc.org/wondershaper/
Stand: 16. April 2002

“MRTG – Multi Router Traffic Grapher”
Tobias Oetiker
http://people.ee.ethz.ch/~oetiker/webtools/mrtg/
Stand: 09. Juni 2003

C Selbständigkeitserklärung

Ich erkläre hiermit, dass ich diese Projektarbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich und sinngemäß aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls das Landesschulamt gemäß dem Gesetz Entzug des aufgrund dieser Projektarbeit verliehenen Zeugnisses berechtigt ist.

Berlin, _____ _____

Unterschrift