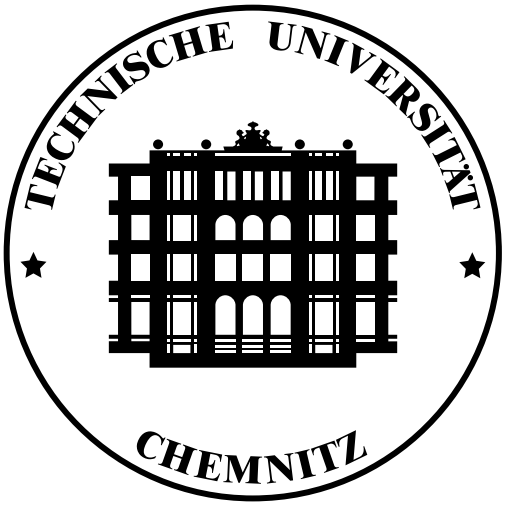
Technische Universität Chemnitz

– Diplomarbeit –

Dynamische Bandbreitenbeschränkung mit QoS



© Jan Horbach

24. September 2001

Bibliographische Bezeichnung:
Jan Horbach: "Dynamische Bandbreitenbeschränkung mit QoS" - 2001 - 74 S., Technische Universität Chemnitz, Fakultät für Informatik, Diplomarbeit, 2001

Kurzreferat:
Diese Diplomarbeit beschäftigt sich mit der "schleichenden Abschaltung" begrenzter Netzzugänge bei Überschreitung eines vorgegebenen Datenvolumens und der Anwendung dieser Technologie im Chemnitzer Studentennetz. Nutzer, die bestimmte Transfervolumina überschreiten, werden schrittweise einer immer schlechter bewerteten Verkehrsklasse zugeordnet, wo ihnen weniger Bandbreite zur Verfügung steht. Dazu werden die Möglichkeiten, die der Linux-Kern hinsichtlich Quality of Service bietet, genutzt.

Danksagung

Der Autor der vorliegenden Arbeit möchte sich beim CSN-Team für die Unterstützung bedanken, insbesondere bei Christian Krause und Mirko Parthey als Hauptansprechpartner, Heiko Jehmlich für die Hilfe bei der Einbindung der Skripte in das bestehende System und Lutz Neugebauer für die Beantwortung aller auf die Datenbank bezogenen Fragen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CBQ	Class-Based Queuing
CoS	Classes of Service
CSN	Chemnitzer Studentennetz
DFN	Deutsches Forschungsnetz
DoS	Denial of Service
FIFO	First-In-First-Out-Warteschlange
GWiN	Gigabit-Wissenschaftsnetz
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IPX	Internet Packet Exchange
NGI	Next Generation Internet
PHP	PHP Hypertext Preprocessor
QoS	Quality of Service
RED	Random Early Detection
SFQ	Stochastic Fair Queuing
SNMP	Simple Network Management Protocol
SSL	Secure Socket Layer
TBF	Token Bucket Filter
TC	Traffic Control
TCP	Transmission Control Protocol
TLS	Transport Layer Security
VLAN	Virtual Local Area Network
WWW	World Wide Web
XML	Extensible Markup Language

1. Aufgabenstellung für die Diplomarbeit

Name, Vorname des Diplomanden: Horbach, Jan
Immatrikulations-Nr.: 17518

Thema: Dynamische Bandbreitenbeschränkung mit QoS

Zielstellung:
Netzzugänge, deren Abrechnungen auf einem limitierten Datenvolumen pro Zeit basieren, haben das Problem, entscheiden zu müssen, was im Falle einer Überschreitung des vorgegebenen Limits geschehen soll.

Die einfachste Lösung ist sicherlich die sofortige Sperrung des Zugangs mit einer vorherigen Verwarnung des betreffenden Administrators. Leider stellt dieses Verfahren aber aus Nutzersicht einen ziemlich harten Eingriff in seine Arbeit dar und wird subjektiv als "unfreundlicher Akt" wahrgenommen.

Die Aufgabe besteht darin, mit Hilfe der unter Linux neu geschaffenen Möglichkeiten des QoS eine "schleichende Abschaltung" des Netzzuganges zu realisieren. Eine Auswertung des Datenverkehrs sorgt bei einer Überschreitung des vorgesehenen Datenvolumens dafür, dass der vom/zum betreffenden Host gehende Netzverkehr in eine immer schlechter bewertete Verkehrsklasse gelangt.

Der Administrator des betreffenden Hosts sollte über diese Maßnahme nach Möglichkeit automatisch informiert werden.

Es soll geprüft werden, inwieweit sich für die Administration des Systems neue Technologien wie XML-RPC eignen.

Die Lösung soll auf ausschließlich freien Softwareprodukten beruhen und so dokumentiert sein, dass weiterführende Arbeiten nach einer kurzen Einarbeitungszeit möglich sind.

Betreuender Hochschullehrer:	***Prof. Dr. U. Hübner***
Fakultät:	Informatik
Professur:	Rechnernetze und verteilte Systeme
Betreuer:	Dr. J. Anders
Beginn am:	01.04.2001
Einzureichen am:	30.09.2001

2. Einleitung

Das Internet wird heutzutage sehr stark genutzt, sowohl privat als auch kommerziell. Verhaltensnormen, die sich herausgebildet hatten, als die Dienste des Internets nur von einer kleinen Menge an Nutzern in Anspruch genommen wurden, gelten nur noch eingeschränkt, da die heutigen Nutzer bereits eine gewisse Selbstverständlichkeit im Umgang mit diesem Medium an den Tag legen und ihren Interessen gemäß benutzen. Doch die verfügbaren Ressourcen sind begrenzt, das Netz ist *over-subscribed*, weshalb ein bewussterer Umgang mit diesen Ressourcen wünschenswert wäre. Kaum einem Nutzer kann die Bandbreite garantiert werden, die ihm durch seine Anbindung theoretisch zur Verfügung steht, sondern nur ein Bruchteil davon. Um einer Überlastung des Netzes auszuweichen, werden die Bandbreiten begrenzt oder Traffic-Limits festgelegt, bei deren Überschreitung entweder der Zugang gesperrt oder der Preis erhöht wird.

Den Universitäten, die am **Gigabit-Wissenschaftsnetz (GWiN)** des **Deutschen Forschungsnetzes (DFN)** angeschlossen sind, steht beispielsweise nur ein bestimmtes Transfervolumen pro Monat zu, was aber durch eine übermäßige Ausnutzung der zur Verfügung stehenden Bandbreite leicht überschreitbar ist. Der TU Chemnitz etwa steht ein Anschluss der Kategorie 11 mit 155 MBit/s und einem monatlichen Transfervolumen von 3000 GB (ab Oktober 2001 die doppelte Menge) zur Verfügung, für den jährlich 550.000 DM Entgelt anfallen. Die nächste Kategorie wäre bei gleicher Bandbreite, aber einem doppelten Datenvolumen um 200.000 DM teurer, bei Überschreitung ist mit Mahnungen und Bandbreitenbeschränkung zu rechnen [HeHo01] [URZ01].

Um dem aus dem Weg zu gehen, ist die Idee, für jeden Nutzer der Hochschule (bzw. einer bestimmten Nutzergruppe) ein Traffic-Limit festzulegen, nach dessen Überschreitung der Netzzugang gesperrt wird (gegebenenfalls erst nach einer Verwarnung bei erstmaligem Überschreiten). Das Limit sollte dabei aber nicht so festgelegt sein, dass das zur Verfügung stehende Transfervolumen einfach auf die Anzahl der Nutzer aufgeteilt wird, da nur einige wenige das Limit tatsächlich ausreizen werden, sondern so, dass hauptsächlich die Nutzer eingeschränkt werden, die das Netz am stärksten beanspruchen. Diese Variante der Beschränkung des Netzverkehrs wird derzeit zum Beispiel im **Chemnitzer Studentennetz (CSN)** eingesetzt.

Aber die Idee eines solchen Limits bringt auch Probleme mit sich. Denn es kann durchaus vorkommen, dass man eine größere Datenmenge benötigt. Ausnahmeregelungen sind zwar denkbar, könnten aber einen nicht unerheblichen Aufwand nach sich ziehen. Weiterhin birgt ein Limit die Gefahr in sich, dass bei Ablauf der Gültigkeitsfrist von den Nutzern noch schnell

das verbleibende Datenvolumen ausgeschöpft wird. Bei Tageslimits wird das zwar nicht so stark in Erscheinung treten, aber bei Wochen- oder gar Monatslimits ist garantiert damit zu rechnen. Außerdem wird die Begrenzung des Datenvolumens von den Nutzern sicher als "unfreundlicher Akt" und Beschneidung der persönlichen Freiheit empfunden.

Der Ausweg wäre eine "schleichende Abschaltung" des Netzzuganges, bei der der Nutzer je nach beanspruchtem Datenvolumen in eine immer schlechter bewertete Verkehrsklasse eingeteilt wird, wo ihm weniger Bandbreite zur Verfügung steht. Das kann soweit gehen, dass er sich letztendlich in einer Klasse befindet, wo es ihm nicht mehr möglich ist, ein bestimmtes Limit zu überschreiten. Das sollte aber erst bei einer wirklich exzessiven Nutzung des Netzes geschehen, die Hochstufung sollte langsam erfolgen, so dass Nutzer, die nur selten eine größere Datenmenge beanspruchen, nur leicht eingeschränkt werden. Auf die Art und Weise stellen dann gelegentliche größere Transfers kein Problem mehr dar, da keine Sperrung erfolgt und die "Behinderung" durch die Beschränkung der Bandbreite in diesen Fällen nur minimal ist. Hält sich der Nutzer nach seiner Hochstufung an bestimmte Datenvolumina, wird er schrittweise wieder einer besseren Verkehrsklasse zugeteilt.

Das System sollte sich weitestgehend selbst regeln, so dass bestimmte Parameter eingehalten werden, wie z.B. eine bestimmte Monatsobergrenze der Trafficmenge und garantierte Mengen pro Nutzer und Monat. Außerdem muss das Netz die ganze Zeit über benutzbar bleiben, die Bandbreiten dürfen nicht so weit begrenzt werden, dass ein vernünftiges Arbeiten nicht mehr möglich ist. Der Vorteil an dieser automatischen Regelung ist die optimale Ausnutzung der vorgegebenen Monatsobergrenze, jedoch ohne diese zu überschreiten. So sind z.B. in Ferienzeiten, in denen weniger Personen das Netz nutzen, höhere Bandbreiten für die einzelnen Verkehrsklassen möglich, so dass der verfügbare Monatstraffic besser aufgeteilt werden kann.

Ein weiterer Vorteil ist der geringere Verwaltungsaufwand: Die Verkehrsklassen müssen nur einmal vor dem Einsatz eingerichtet werden, die Ausstellung von Verwarnungen und die Sperrung von Anschlüssen würden entfallen. In Ferienzeiten o.ä. kann der verfügbare Monatstraffic auf die verbleibenden Nutzer besser aufgeteilt werden, ohne dass für diesen Zeitraum Limits geändert werden müssen.

Mit der "schleichenden Abschaltung" durch eine dynamische Bandbreitenbeschränkung und der automatischen Regelung soll sich diese Arbeit beschäftigen.

3. Grundlagen

3.1. Quality of Service in Linux

3.1.1. Quality of Service

Nach einer Definition, die 1997 auf dem **Next Generation Internet (NGI)** Workshop in Virginia geprägt wurde, beschreibt **Quality of Service (QoS)** die Unterscheidung von verschiedenen Verkehrsklassen und Services sowie deren unterschiedliche Behandlung:

"[...] the group came to the conclusion that the best definition it could formulate for QoS was one that defined methods for differentiating traffic and services - a fairly broad brush stroke, and one that may be interpreted differently." [FeHu98, Seiten xv-xvi].

Unter **Quality** versteht man dabei, dass bestimmte qualitative Parameter eingehalten bzw. garantiert werden, z.B. eine verlässliche Datenlieferung, kein Datenverlust, garantierte Bandbreiten, minimale Verzögerung (geringe **Latenz**) oder konstante Verzögerungscharakteristika (kaum **Jitter**[1]), aber auch die effizienteste Nutzung von Netzwerkressourcen (kürzeste Entfernung, schnellster Pfad) oder die bevorzugte Behandlung von Management-Daten.

Services hingegen legen verschiedene Arten von Anwendungen bzw. Protokollen fest, den sogenannten **Classes of Service (CoS)**. Das können die Unterscheidung von Anwendungsprotokollen wie E-Mail, **World Wide Web (WWW)**, Chat oder Videostreams, aber auch unterschiedliche Protokollumgebungen wie **Internet Protocol (IP)**, **Internet Packet Exchange (IPX)** und AppleTalk sein. Entsprechend können diese Services u.a. anhand des benutzten Protokolls, der Quell- oder Zieladresse bzw. des entsprechenden Ports klassifiziert werden.

[1] Schwankung der Latenzzeit bei Paketen einer Verbindung

3.1.2. Das Werkzeug Traffic Control

Linux unterstützt eine Reihe von QoS-Verfahren: einfache Queuing-Algorithmen, die unterschiedliche Vergabe von Bandbreiten und Prioritäten, aber auch Algorithmen zur Überlastvermeidung von TCP-Datenströmen und zur Verteilung von Traffic auf mehrere Netzinterfaces. Als leistungsfähigster Algorithmus ist das Class-Based Queuing (CBQ) zu nennen, das die Unterteilung des Traffics in verschiedene Klassen ermöglicht.

Die Behandlung erfolgt beim 2.2er Kernel nur für ausgehenden Traffic, erst ab der Version 2.4 können zusätzlich ankommende Pakete erfasst werden. Durch den Einsatz in einem Router lassen sich jedoch auch ankommende Daten begrenzen, indem die Maßnahmen auf dem Interface ergriffen werden, das die Daten in das lokal angeschlossene Netz weiterleitet. Dadurch erfolgt allerdings noch keine Verlangsamung des Datenstroms an sich, sondern nur die Verbindung zwischen Router und lokalem Netz wird hiermit begrenzt. Wegen des Wegwerfens von Paketen wird bei TCP-Strömen allerdings auch kein `ACK` gesendet, so dass der vorgelagerte Router die Rate der zu sendenden Pakete ebenfalls drosselt, da er eine Überlastsituation vermutet [Stev94, Seiten 285-286].

Um QoS unter Linux nutzen zu können, muss der Support im Kernel aktiviert werden (*Networking Options —>QoS and/or fair queueing*). Die benötigten Algorithmen sind als Modul oder fest einkompiliert auszuwählen. Um die benutzte Bandbreite von Datenströmen abzuschätzen, ist der *QoS/Rate Estimator* notwendig. Die Classifier (Filter) dienen zur Klassifizierung der verschiedenen Datenströme und der Zuordnung zu einzelnen Verkehrsklassen.

Das Werkzeug **Traffic Control (TC)** dient zum Management von Queuing Disciplines, Verkehrsklassen und Trafficfiltern. In den nächsten Abschnitten soll dessen Funktion schrittweise erläutert werden.[2]

Wenn man die Queuing Disciplines oder die Classifier als Module kompiliert hat, sind sie vor dem Aufruf von `tc` explizit mit `modprobe` zu laden:

```
modprobe sch_qdisc
modprobe cls_classifier
```

[2] Vergleiche hierzu auch [Horb00a], [Radh99] und [Lamb99].

3.1.3. Zuweisung der Queuing Disciplines

Im ersten Schritt muss die gewünschte Queuing Discipline, d.h. der zum Queuing der Pakete zu verwendende Algorithmus, an das betreffende Netzinterface gebunden werden. Das geschieht mit folgendem Aufruf:

```
tc qdisc add dev DEVICE [handle X:] {root|parent CLASSID}
   [estimator INTERVAL TIME_CONSTANT] QDISC_KIND OPTIONS

  dev         - Netzwerkdevice (Interface)
  handle      - Handle der QDisc (eindeutige ID) der Form X:, z.B. 1:
  root        - Bindung bezieht sich direkt auf das Interface
  parent      - Bindung bezieht sich auf Parent mit CLASSID
  estimator   - Steuerung des Rate Estimator (Zeitraum zwischen Messungen
                und Zeitkonstante zur Mittelwertbildung in Sekunden)
  QDISC_KIND = {cbq|red|sfq|tbf|...}
```

Eine Auswahl der möglichen Queuing Disciplines soll jetzt erläutert werden. Dabei wird vor allem auf diese Algorithmen eingegangen, die im Rahmen dieser Arbeit wahrscheinlich auch eingesetzt werden. Das betrifft vor allem CBQ zur Definition der Verkehrsklassen und TBF zur Beschränkung der Bandbreiten. SFQ könnte verwendet werden, wenn eine unbeschränkte Klasse definiert wird, um die Bandbreite gleichmäßig auf alle Nutzer aufzuteilen. RED wird wahrscheinlich nicht zum Einsatz kommen, es ist hier aber der Vollständigkeit halber mit enthalten, da es sich um einen sehr wichtigen Algorithmus handelt.

Class-Based Queuing

Normalerweise werden zu versendende Pakete in der Reihenfolge ihrer Erzeugung in eine Warteschlange gelegt, über die sie dann in derselben Reihenfolge gesendet werden. Beim **Priority Queuing** werden höherpriorisierte Pakete vor niedrigpriorisierteren in die Output-Queue eingeordnet. Dazu muss jedes Paket analysiert und gegebenenfalls umgeordnet werden, was den Durchsatz verringert. Das **Class-Based Queuing (CBQ)** besitzt demgegenüber eine geringere Latenz. Hier existieren mehrere Output-Queues, auf die die Bandbreite hierarchisch auf Klassen je nach Bedarf verteilt wird. In der Praxis wird das durch die unterschiedliche Vergabe von Ressourcen geregelt.

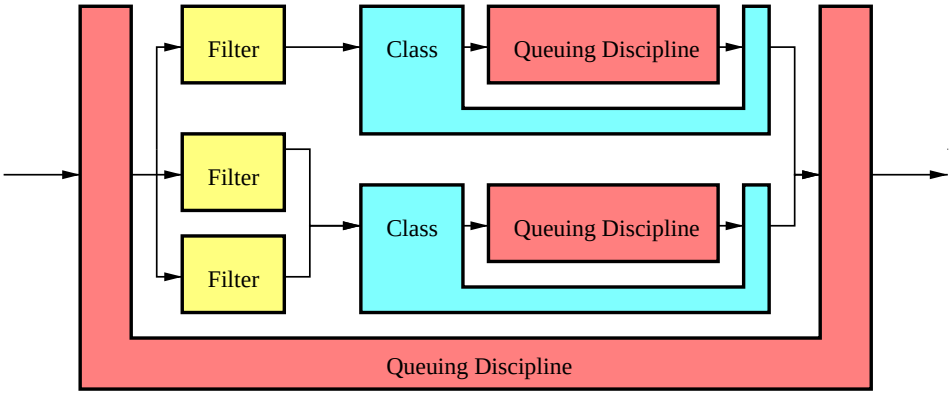


Abbildung 3.1-1.: Einfache Queuing Discipline mit mehreren Klassen [Quelle: AHSK99]

Die Klassen sind in einer Baumstruktur abgelegt, in der sich die Kinder von den Eltern bei Bedarf Bandbreite borgen können, wenn dort noch Kapazität vorhanden ist. Im Gegenzug können die Kinder nicht benötigte Bandbreite den Eltern zur Verfügung stellen, die dann gegebenenfalls an andere Kinder verborgt wird. Den Blättern im Baum können unterschiedliche Queuing Disciplines zugewiesen werden, was CBQ ziemlich leistungsfähig macht, denn so können verschiedene CoS durch geeignete Algorithmen behandelt werden. Wird einem Blatt keine spezielle Queuing Discipline zugewiesen, wird standardmäßig eine **First-In-First-Out-Warteschlange (FIFO)** verwendet. Die Klassifikation erfolgt über Filter: generische Filter (z.B. routingbasiert) oder spezielle Filter (z.B. U32-Classifier). Klassen und Filter werden in den nächsten Abschnitten genauer besprochen.

```
tc qdisc add ... cbq bandwidth BPS avpkt BYTES [mpu BYTES]
                 [cell BYTES] [ ewma LOG ]

  bandwidth - reale Bandbreite des Interfaces
  avpkt     - durchschnittliche Paketgröße
  mpu       - minimale Paketgröße
  cell      - Anzahl Bytes, in der die Transferzeit gemessen wird
  ewma      - Exponentially Weighted Moving-Average (Beeinflussung
              aktueller Werte durch alte Werte bei Messungen)
```

Token Bucket Filter

Der **Token Bucket Filter (TBF)** hat die Aufgabe, die Menge und Rate des Traffics zu kontrollieren, der in das Netz eintritt. Der Bucket wird analog zu einem regelmäßig mit Münzen "gefütterten" Geldbeutel vom System in bestimmten Abständen mit Tokens versorgt, von denen jedes eine gewisse Menge von sendbaren Datenbytes repräsentiert. Daten können nur

gesendet ("gekauft") werden, wenn für diese Menge genügend Tokens im Bucket ("genügend Münzen im Geldbeutel") sind. Dadurch sind auch Bursts möglich: wenn längere Zeit nichts gesendet wurde und sich mehrere Tokens angesammelt haben - analog zum Sparen von Geld für eine größere Ausgabe. Zusätzlich ist eine maximale Burst-Größe festlegbar.

```
tc qdisc add ... tbf limit BYTES buffer BYTES[/BYTES] rate KBPS
                 [mtu BYTES[/BYTES]] [peakrate KBPS] [latency TIME]

  limit        - Größe des TBF
  buffer/burst - Schwelle, bis zu der Bursts gesendet werden dürfen
  mtu          - Maximum Transfer Unit
  rate         - Transferrate
  peakrate     - maximale Transferrate
  latency      - Latenzzeit
```

Random Early Detection

Das **Transmission Control Protocol (TCP)** hat die Eigenschaft, in einer Überlastsituation, d.h. wenn Pakete des Datenstroms verloren gehen, die Datenrate zu drosseln.[3] Wenn viele TCP-Ströme gleichzeitig präsent sind, gehen häufiger Pakete verloren, so dass die Ströme ihre Datenrate ständig wieder herunterfahren, wodurch diese stark schwankt. **Random Early Detection (RED)** verhindert diesen Überlastkollaps durch zufälliges Wegwerfen von Paketen (abhängig vom Füllstand der Queue), um der Flusskontrolle zu ermöglichen, die Datenrate so festzulegen, dass Überlast vermieden wird.

```
tc qdisc add ... red limit BYTES min BYTES max BYTES avpkt BYTES
                 burst PACKETS probability PROB bandwidth KBPS

  min         - minimale Paketgröße
  max         - maximale Paketgröße
  burst       - erlaubte Anzahl von Paketen in einem Burst
  probability - Drop-Wahrscheinlichkeit (Default: 2%)
```

Stochastic Fair Queuing

Mittels **Stochastic Fair Queuing (SFQ)** werden die Datenströme durch eine Hashbildung über Quell- und Zieladresse auf mehrere Queues verteilt, die gleichmäßig per **Round Robin**

[3] Für eine detaillierte Beschreibung siehe [Stev94, Seiten 310-312].

entleert werden. Dadurch kann eine relativ gleichmäßige und faire Vergabe der vorhandenen Bandbreite auf alle Nutzer garantiert werden. Die Hash-Funktion wird in bestimmten Zeitintervallen geändert, um Hash-Kollisionen (verschiedene Verbindungen nutzen dieselbe Warteschlange) zu minimieren.

```
tc qdisc add ... sfq perturb SECS quantum BYTES

  perturb - Zeitintervall, nachdem Hash-Funktion geändert wird
  quantum - Bytes, die pro Runde übertragen werden
```

Anzeige der eingerichteten Queuing Disciplines

Um sich die definierten Queuing Disciplines und Informationen zu behandeltem Traffic anzeigen zu lassen, existiert folgender Aufruf:

```
tc [-s] qdisc {show|ls} dev DEVICE

  -s - statistische Angaben (Anzahl behandelter Pakete etc.)
```

3.1.4. Einrichten der Klassenhierarchie

Falls als Queuing Discipline CBQ verwendet wurde, werden im zweiten Schritt die Klassen für die verschiedenen Traffic-Arten eingerichtet und in einem Baum angeordnet. Beim Anlegen einer Klasse wird dabei immer der Parent-Knoten mit angegeben, um die Verknüpfung herzustellen.

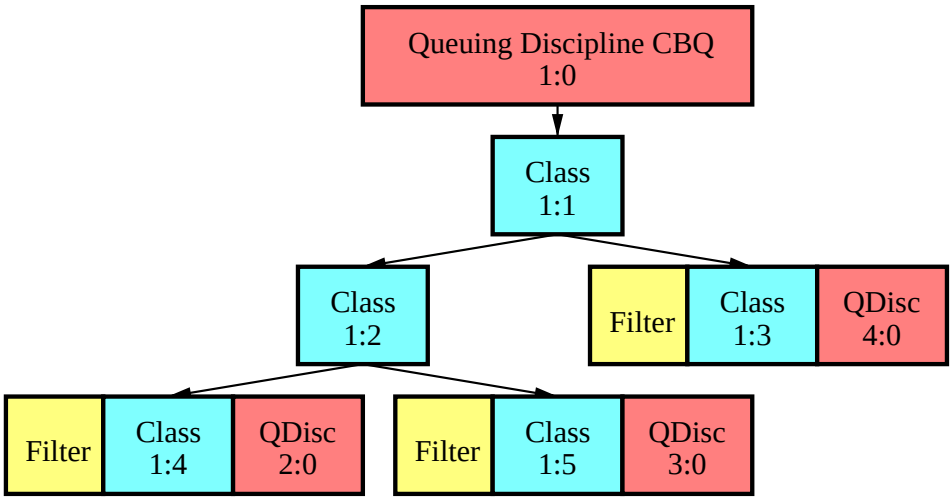


Abbildung 3.1-2.: Klassenbaum mit 3 Blättern und den zugehörigen Filtern

```
tc class add dev DEVICE classid CLASSID {root|parent CLASSID}
   cbq bandwidth BPS rate BPS [avpkt BYTES] [mpu BYTES] [allot BYTES]
   [weight RATE] maxburst PACKETS [minburst PACKETS] [prio NUMBER]
   [cell BYTES] [bounded] [isolated]
   [estimator INTERVAL TIME_CONSTANT]

  classid      - ID der Form X:Y (X = ID der QDisc, Y = ID der Klasse)
  rate         - zugewiesene Bandbreite
  allot        - MTU + Größe des MAC-Headers
  weight       - Gewicht der Klasse (optional, proportional zu "rate")
  min/maxburst - minimale/maximale Anzahl von Paketen in einem Burst
  prio         - Priorität der Klasse (zwischen 1 und 8, 1 = höchste)
  cell         - Anzahl Bytes, bei der Transferzeit gemessen wird
  bounded      - Klasse darf sich keine Bandbreite borgen
  isolated     - Klasse teilt sich keine Bandbreite mit Nicht-Kindern
```

Anzeige der eingerichteten Klassen

Die eingerichteten Klassen und Angaben dazu, wie viele Pakete durch welche Klasse behandelt wurden, wo Bandbreite geborgt wurde usw., können mit folgendem Aufruf angezeigt werden:

```
tc [-s] class {show|ls} dev DEVICE [root|parent CLASSID]

  -s - statistische Angaben (Anzahl behandelter Pakete etc.)
```

3.1.5. Klassifizierung der Pakete

Schließlich müssen nur noch die einzelnen Pakete ihren Klassen (hier als Flows bezeichnet) zugeordnet werden. Dazu wird durch den U32-Classifier über die Match-Anweisungen der Paketkopf ausgewertet, und durch die Flow-ID geschieht die Zuordnung zu einer Klasse. Eine Match-Anweisung bezieht sich auf ein Byte (u8), ein Wort (u16) oder ein Doppelwort (u32) mit entsprechender Bitmaske (max. 0xFF, 0xFFFF oder 0xFFFFFFFF bzw. relevante Bits bei IP-Adressen). Die Filterregeln werden der Reihe nach durchgangen, bis eine der Regeln zutrifft. Ansonsten wird das betreffende Paket nach dem Best-Effort-Prinzip behandelt.

```
tc filter add dev DEVICE [handle X:Y:Z] {root|parent CLASSID}
   [prio PRIO] [protocol ip] [estimator INTERVAL TIME_CONSTANT]
   u32 [match SELECTOR] flowid CLASSID

  handle    - Handle des Filters (optional)
  parent    - Handle der Root-QDisc
  prio/pref - Priorität des Filters
  flowid    - Blatt im Klassenbaum
```

Protokoll	Feld	Typ	Maske	Beschreibung
ip	src/dst	<ipaddr>/BITS		Quell-/Zieladresse
	tos, dsfield,	<u8>	0xFF	TOS-Byte, DSCP bzw.
	precedence	<u8>	0xFF	IP-Precedence
	nofrag, firstfrag,			Bits zur Steuerung
	df, mf			der Paketfragmentierung
	ihl	<u8>	0xFF	Länge des Headers
	protocol	<u8>	0xFF	Layer4-Protokolltyp
	sport/dport	<u16>	0xFFFF	Layer4-Quell-/Zielport
	icmp_type/icmp_code	<u8>	0xFF	ICMP-Typ/Code
udp	src/dst	<u16>	0xFFFF	UDP-Quell-/Zielport
tcp	src/dst	<u16>	0xFFFF	TCP-Quell-/Zielport
icmp	type/code	<u8>	0xFF	ICMP-Typ/Code

Tabelle 3.1-1.: Aufbau eines Selectors [Quelle: Lore00]

Anzeige der eingerichteten Filter

```
tc filter {show|ls} dev DEVICE [root|parent CLASSID]
```

3.1.6. Traffic Control unter Volllast

Die in dieser Arbeit angesprochenen Algorithmen zeigten bei Tests ein sehr gutes Verhalten hinsichtlich Stabilität unter Volllast, aber auch im Langzeitversuch [Horb00a]. QoS lässt sich auch während einer laufenden Übertragung ein- und ausschalten, ohne dass vorhandene Datenströme gestört werden.

Allerdings können nicht beliebig viele Klassen definiert werden. Schon bei 2000 Klassen ist der Overhead, der beim Zuordnen entsteht, deutlich zu spüren: Auf einem PIII-450 dauert es ca. 25min, 5000 Klassen und die zugehörigen Filter anzulegen (bei 2000 Klassen entsprechend weniger), und es werden auf einer 100MBit-Leitung maximal 14-15MBit/s erreicht, da der Rechner nur noch mit dem Filtern und dem Umordnen der Pakete beschäftigt ist. Es ist also nicht möglich, für jeden Nutzer eine Klasse einzurichten, sondern es ist nur eine kleine und festgelegte Anzahl von Verkehrsklassen sinnvoll, in die die Nutzer dann je nach erzeugtem Traffic eingeordnet werden.

Viele Filter sind jedoch möglich, wenn der Rechner, der die Filterung vornimmt, schnell genug ist. Tests auf einem 100MBit-Netzwerk mit zwei angeschlossenen Rechnern haben folgende Werte ergeben:

	K6-200	**Athlon 500**	
Anzahl Filter	**Kernel 2.2.16**	**Kernel 2.2.18**	**Kernel 2.4.2**
ohne QoS	79,70	77,49	77,85
mit 250 Filtern	50,75	76,98	76,77
mit 500 Filtern	41,17	76,77	76,38
mit 1000 Filtern	27,96	78,24	76,35

Tabelle 3.1-2.: Maximale Datenraten mit vielen Filtern in MBit/s

Wie man sieht, ist der K6 mit der Filterung stark überfordert, während auf dem Athlon praktisch keine Auswirkungen festzustellen sind. Da beim Einsatz im CSN nicht mit mehr als 500 Filtern gerechnet werden sollte, dürfte der dort benutzte Rechner, ein PII-400, ausreichend sein.

3.2. XML-RPC

3.2.1. Was ist XML-RPC?

XML-RPC [User01] ist ein Remote-Procedure-Call-Protokoll, welches auf der Basis der **Extensible Markup Language (XML)** und dem **Hypertext Transfer Protocol (HTTP)** realisiert ist.[4] HTTP dient dabei als Übertragungsprotokoll, welches die XML-kodierten Nutzdaten transportiert.

XML-RPC ist ein sehr einfaches Protokoll, d.h. es ist einfach zu verstehen und einfach zu implementieren. Der große Vorteil ist seine Sprachunabhängigkeit, so dass Daten zwischen Applikationen in verschiedenen Programmiersprachen ausgetauscht werden können. So existieren derzeit Beispielimplementationen in Java, Perl, Tcl, Python u.a. Zur sicheren Übertragung können die HTTP-Mechanismen wie **Secure Socket Layer (SSL)** bzw. **Transport Layer Security (TLS)** und HTTP Authentication genutzt werden. Die Übergabemöglichkeit strukturierter Daten (Bäume, Listen etc.) macht es im Webumfeld auch als CGI-Ersatz interessant. Weiterhin kann es bei Management-Aufgaben auch als Alternative zum **Simple Network Management Protocol (SNMP)** benutzt werden.

[4] Siehe auch [Horb00b].

3.2.2. Datentypen in XML-RPC

In XML-RPC sind mehrere Datentypen definiert, sowohl einfache Skalare als auch zusammengesetzte Datentypen: Felder und Tupel. In ihnen ist jeder Datentyp als Wert erlaubt, auch wieder Felder und Tupel, womit komplexere Datenstrukturen möglich werden.

Skalare Datentypen

Datentyp / Tag	Bedeutung	Beispiel
`<i4>` oder `<int>`	4-Byte signed Integer	`47`
`<boolean>`	Wahrheitswert 0 (false) oder 1 (true)	`1`
`<string>` oder n.a.	Zeichenkette	`Hello World!`
`<double>`	Gleitkommazahl	`39.1`
`<dateTime.iso8601>`	Datum-Zeit-Angabe	`19760906T09:45:00`
`<base64>`	Base64-kodierte Binärdaten	`bml4Cg==`

Tabelle 3.2-1.: Übersicht skalarer Datentypen

Felder (Arrays)

Felder bestehen aus einer Liste von Werten, wobei im Gegensatz zu "normalen" Feldern die Datentypen gemischt verkommen dürfen:

```
<array><data>
  <value>Wert</value>
  ...
</data></array>
```

Tupel (Structs)

Tupel bestehen aus einer Liste von Name-Wert-Paaren. Ein Name ist dabei eine beliebige Zeichenkette, als Wert kann wieder jeder Datentyp eingesetzt werden:

```
<struct>
  <member>
    <name>Zeichenkette</name>
    <value>Wert</value>
  </member>
  ...
</struct>
```

3.2.3. Methodenaufruf

Der Body des Requests (**Payload**) ist von `<methodCall>... </methodCall>` eingeschlossen, um einen Methodenaufruf kenntlich zu machen. Innerhalb dieser Tags muss es einen `<methodName>... </methodName>` geben, dessen Inhalt (der Name der Methode) nur aus Buchstaben, Zahlen, "_", ".", ":" und "/" bestehen darf. Die Interpretation des Namens ist jedoch dem Server überlassen, er kann z.B. als eine Objektmethode oder ein File aufgefasst werden. In der verwendeten PHP-Implementation wird beispielsweise ein Mapping von Methodennamen auf PHP-Funktionen verwendet. Wenn Parameter übergeben werden sollen, erfolgt noch die Angabe von `<params>... </params>`, innerhalb derer mehrere Parameter mit `<param><value>... </value></param>` angegeben werden können.

Zusätzlich werden noch bestimmte Header-Informationen benötigt:

- der Request-Typ: `POST /path/to/rpc-handler HTTP/1.0`
- der `User-Agent` und `Host` müssen angegeben werden
- ebenso der `Content-type: text/xml` und die korrekte `Content-length`

Request: examples.getStateName(47)

```
POST /RPC2 HTTP/1.0
User-Agent: telnet
Host: mykonos.informatik.tu-chemnitz.de
Content-type: text/xml
Content-length: 186

<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>47</i4></value>
    </param>
  </params>
</methodCall>
```

3.2.4. Antwort und Rückgabewerte

Der HTTP-Returncode ist immer `200 OK`, es sei denn, es ist ein Fehler beim HTTP-Server und nicht bei der XML-RPC-Anwendung aufgetreten. Der `Content-type: text/xml` und die korrekte `Content-length` werden ebenfalls zurückgeliefert.

Positive Antwort: Washington

Der Body ist von `<methodResponse>... </methodResponse>` eingeschlossen und enthält `<params>... </params>`, innerhalb derer genau ein `<param><value>... </value></param>` steht:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 139
Content-Type: text/xml
Date: Sun, 20 Feb 2000 10:18:57 GMT
Server: UserLand Frontier/6.2a8-WinNT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>Washington</value>
    </param>
  </params>
</methodResponse>
```

Negative Antwort: Too many parameters

Bei Fehlern wird ein `<fault><value>... </value></fault>` zurückgegeben, in dem sich ein `<struct>... </struct>` mit `faultCode` und `faultString` befindet. Der `faultCode` dient vor allem zur programmtechnischen Auswertung, während der `faultString` für eine menschenlesbare Fehlermeldung gedacht ist. Es existieren allerdings keine globalen Listen für Fehlercodes, stattdessen sind sie von der konkreten Implementation abhängig.

```
...
<methodResponse>
  <fault><value><struct>
    <member>
      <name>faultCode</name>
      <value><int>4</int></value>
    </member>
    <member>
      <name>faultString</name>
      <value>
        <string>Can’t call "getStateName" because there
        are too many parameters.</string>
      </value>
    </member>
  </struct></value></fault>
</methodResponse>
```

3.3. Regelung und Steuerung

Nach DIN 19226 kennzeichnet eine **Regelung**, dass "[...] die Werte der gesteuerten Größen fortlaufend mit den Werten der zugeordneten Führungsgrößen verglichen werden, um trotz einwirkender Störgrößen die Werte der gesteuerten Größen denen der Führungsgrößen anzugleichen." [ToBe89, Seite 201] Kennzeichnend ist dabei ein geschlossener Wirkungsablauf, bei dem sich die Regelgröße ständig selbst beeinflusst [Stro98, Seite 23]:

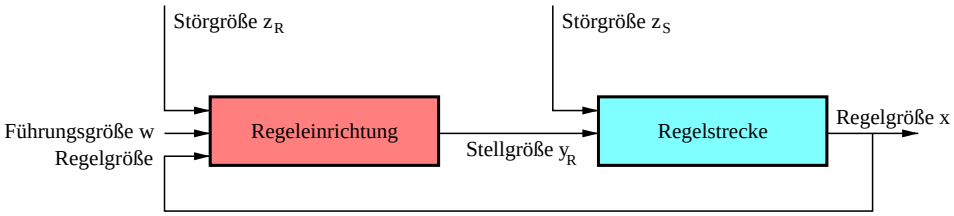


Abbildung 3.3-1.: Aufbau eines Regelkreises [Quelle: Stro98, Seite 23]

Bei einer Steuerung hingegen ist ein offener Wirkungsweg charakteristisch, d.h. die beeinflusste Ausgangs- oder Aufgabengröße wirkt nicht direkt wieder auf die Eingangsgröße zurück. Stattdessen wird die Eingangsgröße neu gemessen, weshalb man sie auch als Messgröße bezeichnet. Außerdem muss auch keine kontinuierliche Rückwirkung erfolgen:

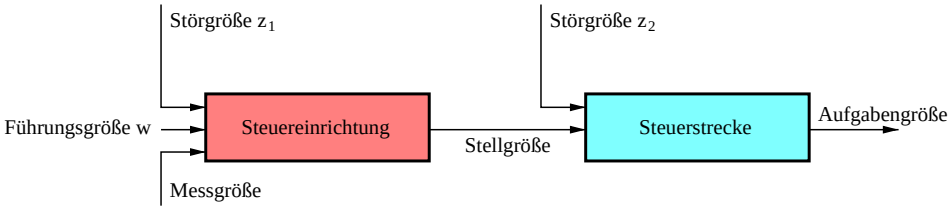


Abbildung 3.3-2.: Aufbau einer Steuerkette [Quelle: Stro98, Seite 24]

Eine **Regeleinrichtung** hat folgende Aufgaben (nach [ToBe89]):

- **Messen** der Regelgröße
- **Vergleichen** der Regelgröße mit der Führungsgröße und Bildung der Regelabweichung
- Bildung der Ausgangsgröße aus der Regelabweichung nach einem bestimmten Algorithmus (**Rechnen**)
- **Verstärken** der Ausgangsgröße, falls die übertragene Energie für die Betätigung des Stellglieds nicht ausreicht

- Betätigung des Stellglieds (**Stellen**)
- **Anzeigen** der Größen sowie Eingabe von Sollwerten und Führungsgrößen (**Bedienen**)

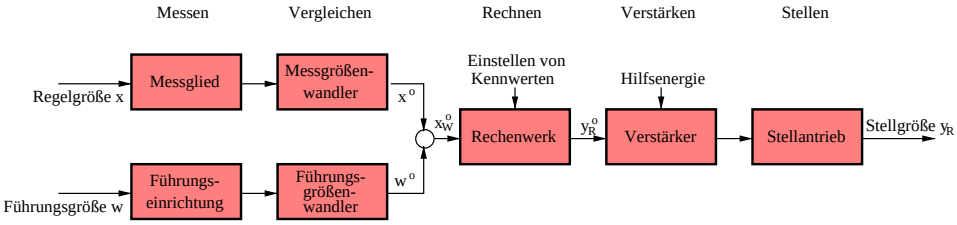


Abbildung 3.3-3.: Aufbau einer Regeleinrichtung [Quelle: ToBe89, Seite 202]

Die Führungsgröße kann in Abhängigkeit von Prozessgrößen gebildet werden (Führungsregelung) oder als Sollwert (Festwertregelung) bzw. Werteverlauf (Zeitplanregelung) vorliegen. [ToBe89]

Für eine Steuerung gelten die hier aufgeführten Angaben analog.

4. Dynamische Bandbreitenbeschränkung

4.1. Prinzipielle Funktionsweise

Zur Realisierung einer dynamischen Bandbreitenbeschränkung werden die Nutzer in Abhängigkeit ihrer verursachten Trafficmenge in Klassen eingeteilt. Standardmäßig werden sie einer unbegrenzten und hochpriorisierten Klasse zugewiesen (oder aus Performancegründen als Best-Effort-Traffic behandelt). Sollten sie bestimmte Trafficgrenzen überschreiten, werden sie am nächsten Tag einer schlechter bewerteten Klasse zugeteilt. Dadurch können die Nutzer ihre Downloads noch mit der alten Geschwindigkeit beenden und müssen erst danach mit den Konsequenzen rechnen. Nur die Extremfälle sollten sofort (stündlich) begrenzt werden, damit sie keinen Schaden anrichten können. Nach einer längeren Einhaltung der Trafficgrenzen werden die Nutzer wieder schrittweise einer besser bewerteten Klasse zugewiesen. Wo die Grenzen der Klassen liegen und wann ein Extremfall eintritt, soll diese Arbeit aufzeigen.

Anders als in [Naum97] sollen die Nutzer nicht von vornherein in bestimmte Klassen eingeteilt werden, sondern alle gelten am Anfang als gleichberechtigt. Zurückstufungen werden nur aufgrund von Limitüberschreitungen vorgenommen.[1]

Zusätzlich ließe sich eine Klasse für internen Traffic von und zum Rechenzentrum der Universität definieren, damit dieser Datenverkehr nicht beschränkt wird. Dadurch können auch Anreize für die Nutzer geschaffen werden, interne Ressourcen wie z.B. den FTP-Server des Rechenzentrums stärker zu nutzen, anstatt sich bestimmte Software von außerhalb der Universität zu besorgen. Allerdings birgt die Einführung einer solchen Klasse auch die Gefahr in sich, dass große Datenmengen dann von Poolrechnern aus gezogen werden und der Transfer dieser Daten ins CSN nur noch intern und unbegrenzt stattfindet. Deshalb ist es erforderlich, dass Traffic zwischen der Universität und dem CSN in das Accounting mit einfließt, um so etwas erfassen zu können. Alternativ bräuchte man auch nicht den gesamten internen Traffic freischalten, sondern nur zu bestimmten angebotenen Services (FTP-Server), damit der Anreiz zur Nutzung interner Ressourcen erhalten bleibt, aber z.B. der Zugang zu den Homeverzeichnissen im Rechenzentrum ebenso beschränkt wird wie der externe Traffic.

Um nutzerbezogene Daten zur Ausnutzung des Netzes zu sammeln, wird ein System benötigt,

[1] Detaillierte Beschreibungen eines differenzierteren Servicemodells und Preisfestlegungen für die unterschiedlichen Services finden sich u.a. in [CSEZ93] und [Shen94]. Das soll jedoch nicht Gegenstand dieser Arbeit sein.

was ein **Accounting** der verursachten Trafficmengen durchführt. Solch ein System nennt man **Meter**:

"A Meter is a process which examines a stream of packets on a communications medium or between a pair of media. The meter records aggregate counts of packets belonging to Flows between communicating entities (hosts/processes or aggregations of communicating hosts (domains)). The assignment of packets to flows may be done by executing a series of rules. Meters can reasonably be implemented in any of three environments - dedicated monitors, in routers or in general-purpose systems." [MHR91]

Meter können in Endsystemen oder in Routern eingesetzt werden. Router eignen sich aus folgenden Gründen jedoch besser dafür [MHR91]:

- Minimierung von Kosten und Overhead durch Konzentration des Accountings auf einen Rechner
- Traffic Control zur Ergreifung von Maßnahmen, z.B. bei Überschreitung von Limits
- Überwachung des Traffics in angrenzenden Netzen (das spielt jedoch für diese Arbeit keine Rolle)

Im CSN existiert bereits ein Meter, der auf dem Router zwischen dem CSN und dem URZ eingesetzt wird und derzeit über `ipchains` realisiert ist, der Umstieg auf `iptables` ist aber geplant. Die dynamischen Bandbreitenbeschränkungen entsprechen dem zweiten Punkt und sollen ebenfalls auf diesem Rechner laufen, und zwar auf dem Interface, welches zum CSN führt.

4.2. Anforderungen und Voraussetzungen

An das System, das die dynamischen Bandbreitenbeschränkungen durchführt, existieren eine Reihe von Anforderungen:

Es soll eine feste Anzahl von Klassen besitzen, deren Parameter über ein Webinterface einstellbar sind. Abgesehen davon soll sich der *DynShaper* allerdings weitestgehend selbst regeln, so dass bestimmte Parameter eingehalten werden: Eine gewisse Monatsobergrenze darf nicht überschritten werden (CSN: ca. 1200 GByte, ab Oktober 2001 eventuell mehr), aber trotzdem muss eine bestimmte Trafficmenge pro Nutzer garantiert sein (CSN: knapp 1 GByte/Monat, ab Oktober 2001 entsprechend mehr). Schöpft ein Nutzer diese ihm zur Verfügung stehende Menge nicht aus, kann der verbleibende Rest an andere Nutzer quasi "verborgt" werden, so dass diese auch mehr Traffic verursachen können. Dabei ist zu beachten, dass ein Nutzer auch mehrere IP-Adressen besitzen kann, aber dass ihm trotzdem nicht mehr zur Verfügung stehen darf als den anderen Nutzern.

Ausnahmeregelungen von diesem Verfahren sollen zumindest prinzipiell möglich sein, so dass zum Beispiel bestimmte Protokolle, Quell- oder Zieladressen gesondert behandelt werden können. Im CSN soll beispielsweise ICMP-Traffic in beide Richtungen begrenzt werden, um **Denial of Service (DoS)** Attacken von außerhalb der Universität vorzubeugen. Eigentlich muss generell nur incoming Traffic berücksichtigt werden, denn nur der wird im DFN wirklich bezahlt, aber das System soll es trotzdem ermöglichen, auch outgoing Traffic zu beschränken.

Das Webinterface soll sich in zwei Teile gliedern: eines für die Administratoren, wo über eine Authentifizierung sichergestellt werden soll, dass diese Aufgabe nur bestimmte Personen übernehmen können, und ein anderes für alle Nutzer, wo diese ihre aktuelle Trafficmenge, ihre Klasse und die zugehörige Bandbreite abfragen können. Bei festen Limits ist so ein Nutzer-Interface nicht zu empfehlen, da dort die Gefahr einer Ausnutzung dieser Limits größer ist, wenn die Nutzer ihren aktuellen Trafficstand abfragen können und sehen, wie viel ihnen noch "zusteht", bevor sie ihr Limit überschreiten. Die Einführung einer solchen Übersicht beim CSN hat diese Befürchtung bestätigt. Auch beim Nutzer-Webinterface muss eine Authentifizierung stattfinden, damit jeder Nutzer nur seine eigenen Daten abfragen kann.

Weiterhin sollte das System relativ transparent für alle Nutzer sein, damit diese jederzeit nachvollziehen können, in welcher Verkehrsklasse sie sich befinden und warum. Dadurch kann die Akzeptanz sichergestellt oder zumindest erhöht werden. Zu diesem Zweck dient einerseits das Webinterface, mit dem jeder Nutzer seine Werte einsehen kann, und zum anderen die Benachrichtigung per E-Mail bei der Einordnung in eine schlechter bewertete Klasse. Auch durch eine einfache und leicht verständliche Struktur des Systems kann die Akzeptanz sicherlich vergrößert werden.

Im Vorfeld des Einsatzes muss Zahlenmaterial zur statistischen Auswertung vorliegen, um damit die Parameter der einzelnen Klassen festlegen zu können, damit die obigen Anforderungen erfüllbar sind. Für den laufenden Betrieb ist es erforderlich, dass QoS im Linux-Kern einkompiliert und die benötigte Software vorhanden ist. Weiterhin muss ein Zugriff auf die aktuellen Trafficdaten in Form einer Datenbank oder von Logfiles gegeben sein.

4.3. Automatische Regelung

Die automatische Regelung der Bandbreiten dient zur optimalen Ausnutzung des vorgegebenen Monatslimits, ohne dieses zu überschreiten. Sie ist streng genommen keine echte Regelung, sondern vielmehr eine Steuerung, da keine kontinuierliche und keine direkte Rückführung der Regelgröße auf die Eingangsgröße erfolgt. Stattdessen wird nur die Auswirkung der Regelgröße gemessen und als neue Eingangsgröße verwendet. Für diese Arbeit soll aber trotzdem der Begriff der Regelung beibehalten werden, da sich eine Steuerung ansonsten nicht von einer Regelung unterscheidet. Die Zuordnung zu Begriffen der Regelung geschieht folgendermaßen:

Regelstrecke	Netzwerk
Regelgröße x	Gesamttraffic pro Monat
Führungsgröße w	Klassenparameter, Bewertungsfunktionen
Regelabweichung x_W	auf Monat hochgerechneter Gesamttraffic
Störgröße z	Nutzerverhalten
Stellgröße y	Bandbreiten der Klassen
Regler	Evaluator
Messglied	Meter
Stellglied	Traffic Control

Tabelle 4.3-1.: Zuordnung zu Regelungsbegriffen (nach [ToBe89] und [Naum97])

Zur Regelung der Bandbreiten wird der bisher in diesem Monat verursachte Traffic auf den gesamten Monat hochgerechnet. Das erfolgt über den Quotienten aus bereits vergangenen Tagen und Gesamtanzahl der Tage des Monats:

$$t_m = t \cdot \frac{d_m}{d} \tag{4.1}$$

wobei t_m der gesamte Monatstraffic, t der bisherige Traffic, d_m die Anzahl der Tage im Monat und d die bisher verstrichenen Tage darstellen.

Übersteigt dieser Wert das Monatslimit, werden die Bandbreiten mit dem Quotienten aus Monatslimit und Monatstraffic multipliziert, was eine Anpassung nach unten ergibt:

$$b_{neu} = b_{alt} \cdot \frac{l_m}{t_m} \tag{4.2}$$

wobei b die Bandbreite und l_m das Monatslimit sind.

Ist das Monatslimit dagegen noch nicht erreicht (es stehen also noch Kapazitäten zur Verfügung), werden die Bandbreiten nach oben korrigiert. Das geschieht diesmal über die Quadratwurzel des o.a. Quotienten, damit die Anpassung nach oben etwas verhaltener abläuft. Ansonsten würden die Bandbreiten vermutlich ständig hin- und herschwanken:

$$b_{neu} = b_{alt} \cdot \sqrt{\frac{l_m}{t_m}} \tag{4.3}$$

Die Regelung erfolgt erst nach den ersten fünf Tagen eines Monats, da die Werte zu Anfang noch zu starken Schwankungen unterworfen sind, als dass eine zuverlässige Hochrechnung möglich ist. Weiterhin werden Korrekturfaktoren nahe 1 nicht berücksichtigt, da sich die Bandbreiten ansonsten ständig ändern.

4.4. Architektur des Systems

4.4.1. Beschreibung der Komponenten

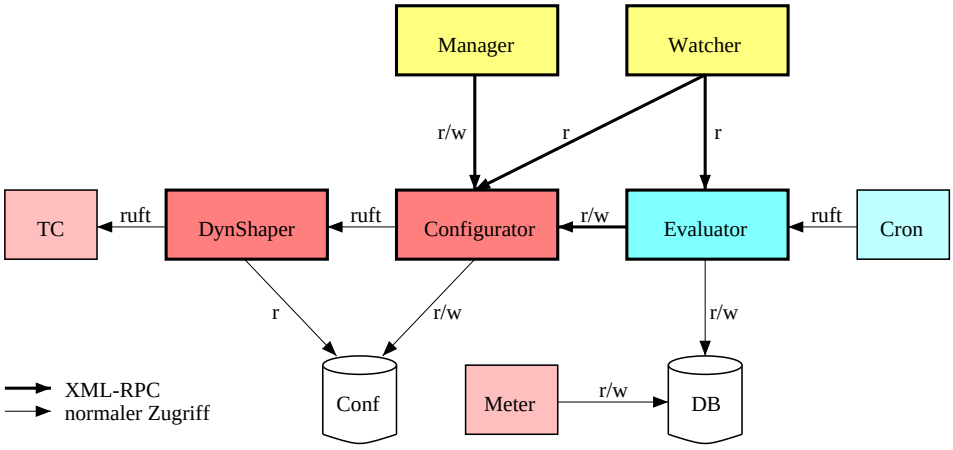


Abbildung 4.4-1.: Architektur des Systems

Durch die Verwendung von XML-RPC kann das System auf mehrere Rechner verteilt werden (im Bild durch unterschiedliche Farben gekennzeichnet). Die heller gekennzeichneten Komponenten sind nicht Bestandteil dieser Arbeit, sondern bereits vorhandene Werkzeuge.

Manager (dsmanager.php)

Der *DynShaper Manager* dient zur Verwaltung des *DynShapers*. Er legt die Parameter für die verwendeten Interfaces, Pfadnamen zu Programmen und die Monatsobergrenze fest, stellt die Werte für die einzelnen Klassen ein und definiert Ausnahmeregelungen. Die Einstellungen sollen nach Möglichkeit nur am Anfang vorgenommen werden müssen, im laufenden Betrieb sollte sich das System soweit wie möglich selbst regeln.

Watcher (dswatcher.php)

Mit dem *DynShaper Watcher* ist es den einzelnen Nutzern möglich, ihre aktuellen Werte wie den verursachten Traffic, die Verkehrsklasse, der sie momentan zugeordnet sind sowie deren Parameter usw. abzufragen. Dabei wird es sich wahrscheinlich nur um eine Referenzimplementation handeln, beim Einsatz wird die Nutzerabfrage sicherlich in die bereits existierenden Programme integriert werden.

Configurator (dsconfigurator.php)

Durch den *DynShaper Configurator* erfolgt die Konfiguration des *DynShapers*. Er verwaltet die Konfigurationsdatei und beantwortet XML-RPC-Anfragen, die Konfigurationsoptionen lesen oder schreiben wollen. Genutzt wird der *Configurator* von den beiden bereits genannten Programmen und vom *Evaluator*.

In der Konfigurationsdatei werden folgende Einstellungen gespeichert:

- internes und externes Interface mit ihren Bandbreiten und Gewichten
- das Monatslimit und ob eine automatische Regelung stattfinden soll
- ab wann eine E-Mail bei Einordnung in eine schlechtere Klasse geschickt werden soll
- Pfade zum *DynShaper, TC, ModProbe* und zum Logfile
- die einzelnen Klassen mit Bandbreite (Rate), Gewicht, Limit, Tagen und IP-Adressen sowie der Festlegung, ob incoming und/oder outgoing Traffic beschränkt werden und ob diese Beschränkung sofort erfolgen soll (bei Extremfällen)
- die Ausnahmen mit Bandbreite (Rate), Gewicht, Match-Anweisungen für incoming und outgoing Traffic sowie der Festlegung, ob der Traffic hart begrenzt werden soll (bounded)

  Vollständige Konfigurationsdatei (Siehe: A.1)

Evaluator (dsevaluator.php)

Der *DynShaper Evaluator* ist dafür zuständig, das Verkehrsaufkommen der Nutzer auszuwerten und die Nutzer bei Bedarf einer anderen Verkehrsklasse zuzuordnen. Dazu wird einmal pro Stunde (aber möglichst versetzt zur Aktualisierung der Traffic-Datenbank durch das CSN) eine Abfrage der aktuellen Werte durchgeführt. Übersteigt der Traffic eines Nutzers bestimmte Grenzen, wird dieser Nutzer am nächsten Tag in eine schlechter bewertete Klasse eingeordnet, in der er eine gewisse Anzahl von Tagen verbleibt, ehe er schrittweise wieder zurückgestuft wird. Treten in dieser Zeit weitere Überschreitungen auf, erhöhen sich die Klasse und die Anzahl der Tage entsprechend. Nutzer mit extremen Verkehrsaufkommen werden sofort der schlechtesten Klasse zugewiesen, damit sie keinen weiteren Schaden anrichten können. Die Klasse und das Datum der Einordnung werden in der Datenbank gespeichert.

Weiterhin dient der *Evaluator* zur automatischen Regelung der Bandbreiten für die Klassen, damit das festgelegte Monatslimit nicht überschritten werden kann.

DynShaper (dynshaper)

Der *DynShaper* dient zur eigentlichen Begrenzung des Traffics. Er wertet die vom *Configurator* erzeugte Konfiguration aus und legt Queuing Disciplines, Klassen und Filterregeln fest, wozu das Werkzeug *TC* benutzt wird. Das Skript kann vom *Configurator* und über diesen auch vom *Evaluator* neugestartet werden, wenn sich Einstellungen geändert haben. Weiterhin sollte es als Startup-Skript (unter `/etc/rc.d/init.d/` o.ä.) vorliegen, um nach einem Neustart die Beschränkungen sofort wieder verfügbar zu haben.

Festlegung der Ausnahmen (Siehe: A.2)

Definition der Klassen (Siehe: A.3)

4.4.2. Angebotene XML-RPC-Funktionen

Funktionen des Configurators

`dynshaper.load(card, target)`
Die Funktion `dynshaper.load` wird vom *Manager* und vom *Evaluator* verwendet. Sie dient dazu, Werte aus der Konfigurationsdatei für die übergebene Registerkarte (Allgemein, Klassen, Ausnahmen) und das übergebene Target (Klasse bzw. Ausnahme) zurückzuliefern. Diese Werte werden in einem Tupel zurückgegeben.[2]

`dynshaper.store(values)`
Zum Speichern und auch zum Löschen der Werte einer Registerkarte vom *Manager* aus dient `dynshaper.store`. Der übergebene Parameter ist ein Tupel aus den zu speichernden Werten und enthält auch die betreffende Registerkarte und das Target. Zurückgegeben wird ein Wahrheitswert, der anzeigt, ob die Aktion erfolgreich war.

`dynshaper.adjust(classes)`
Die Funktion `dynshaper.adjust` dient dem *Evaluator* dazu, alle Klassen komplett auf einmal abspeichern zu können. Das ist vor allem nach einer Regelung der Bandbreiten sinnvoll, da sich hier beim normalen Speichern durch das Sortieren der Klassen nach ihrer (jetzt geänderten) Bandbreite die Reihenfolge ändern könnte, weil die Klassen nacheinander abgespeichert und einsortiert werden, was beim Speichern der nächsten Klasse Probleme bereiten würde, da die Klassen anhand ihrer Nummer identifiziert werden und somit Daten überschrieben werden könnten.
Andererseits ist durch diese Funktion auch ein Geschwindigkeitsvorteil zu erwarten, da alle Klassen gleichzeitig abgespeichert werden statt mehrmals `dynshaper.store` aufzurufen. Übergeben wird ein Feld aus Tupeln mit den Werten der einzelnen Klassen, als Rückgabeparameter wird analog zum normalen Speichern wieder ein Wahrheitswert verwendet.

`dynshaper.class(class)`
Durch `dynshaper.class` kann der *Watcher* die Werte der übergebenen Klasse abfragen. Zurückgegeben wird ein Tupel aus der Klasse selbst, dem Limit dieser Klasse, wie viele Tage man dort verbleibt sowie der Bandbreite, die den Nutzern dort zur Verfügung steht.

`dynshaper.control(command)`
Mit `dynshaper.control` kann dem *DynShaper* ein Kommando übergeben werden. Möglich sind hierbei `start` und `stop`, die den *DynShaper* starten bzw. beenden, `restart` zum Neustart, damit die geänderten Konfigurationsdaten eingelesen werden, sowie `list` und `status` zur Abfrage von Statusinformationen. Genutzt wird diese Funktion vom *Manager* und vom *Evaluator*. Die Rückgabe besteht aus der Ausgabe des *DynShapers*.

[2] Welche Werte das im Einzelnen sind, siehe vorhergehender Abschnitt.

Funktionen des Evaluators

`dynshaper.user(user)`

Die Funktion `dynshaper.user` wird vom *Watcher* verwendet und gibt Werte über einen Nutzer zurück: die IP-Adressen des Nutzers, seinen Tagestraffic, die Klasse, in die er eingeordnet ist und wie viele Tage er sich schon in dieser Klasse befindet. Als Parameter wird das Login des Nutzers übergeben, der Rückgabewert ist ein Tupel aus den obigen Werten und dem Login.

4.4.3. Eingesetzte Software

Zur Implementierung der einzelnen Komponenten wird der **PHP Hypertext Preprocessor (PHP)** [PHPG01] auf einem Apache-Webserver verwendet, da PHP gegenüber der herkömmlichen CGI-Programmierung leichter zu handhaben ist und es weiterhin eine einfach einzubindende XML-RPC-Unterstützung für PHP gibt [Usef00], die zum Beispiel gegenüber der Perl-Variante keine weiteren Module benötigt. Damit XML-RPC funktioniert, muss PHP allerdings mindestens in der Version 3.0.12 vorliegen.

Des Weiteren ist die Serverseite aufgrund der Einbindung von PHP als Apache-Modul leicht realisierbar: Es kann einfach das entsprechende PHP-Skript über den Webserver aufgerufen und die XML-RPC-Daten per `POST` übergeben werden, ohne dass ein extra Server gestartet werden muss, was sich auch günstig auf die Stabilität des Systems auswirkt, da es keine eigenen Prozesse gibt, die ständig laufen müssen und ausfallen könnten. Außerdem wird von PHP auch eine einfache Möglichkeit geboten, um ohne zusätzliche Module auf Datenbanken zugreifen zu können. Für den *Evaluator* wird zusätzlich ein unabhängiger PHP-Interpreter benötigt, der mit XML- und Datenbankunterstützung (CSN: *PostgreSQL*) kompiliert sein muss:

```
./configure --with-pgsql[=DIR] --with-xml
            --with-config-file-path=/path/to/php.ini
make &&  make install
```

Das eigentliche Shaperskript (*DynShaper*) ist ein einfaches (Bash-)Shellskript, da hier nur wenige Konfigurationsoptionen eingelesen werden müssen, die zum Einrichten der Klassen und Filter mit Hilfe des Werkzeugs *TC* notwendig sind, was ebenfalls durch dieses Skript erfolgt. Weiterhin ist ein Shellskript für den Einsatz als Startup-Skript (unter `/etc/rc.d/init.d/` o.ä.) am besten geeignet.

XML-RPC dient zum Austausch der Daten zwischen den einzelnen Programmen des Systems, weil darüber auch komplexere Daten relativ komfortabel übergeben werden können. Außerdem hat die Übertragung über HTTP-`POST` und XML Vorteile gegenüber der Nutzung von SNMP, da einerseits ein bereits vorhandener Webserver genutzt werden kann und kein SNMP-Daemon gestartet werden muss, und zum anderen die Daten auch in einer menschenlesbaren Form vorliegen, wodurch auch von anderer Stelle der *Configurator* bedient werden kann (dabei darf natürlich die Authentifizierung nicht vergessen werden). Außerdem können so die Daten und Konfigurationsoptionen von einer bzw. zwei zentralen Stellen verwaltet werden. Das ist auch einer der Gründe, warum überhaupt XML-RPC eingesetzt werden soll und nicht einfach der *Manager* die Aufgabe des *Configurators* mit übernimmt. Eine andere denkbare Variante ist die mögliche räumliche Trennung auf zwei Rechner, um den Rechner, der die Begrenzung vornimmt, zu entlasten und nicht unnötig mit Nutzeranfragen zu "belästigen".

Als Datenbanksystem wird im CSN schon seit einiger Zeit *PostgreSQL* [PSQL01] eingesetzt und vom *DynShaper*-System mitverwendet.

Entwicklungsumgebung

Zum Vergleich soll die Entwicklungsumgebung des *DynShapers* aufgeführt werden:

- AMD K6-200 mit 160MB RAM
- SuSE Linux 6.2 mit Kernel 2.2.16 und Update auf `glibc` 2.2
- Apache 1.3.12 mit `mod_php` 3.0.15, PHP 3.0.15
- Useful Information Company XML-RPC 1beta9 für PHP
- `libxml` 1.8 und `expat` 1.2

Zusätzlich stand für die Tests noch ein AMD Athlon 500 mit Kernel 2.2.18 und 2.4.2 sowie ein lokales 100MBit-Netzwerk zwischen den beiden Rechnern zur Verfügung.

4.4.4. Bedienung des DynShaper Managers

Der *DynShaper Manager* gliedert sich in drei Teile oder "Registerkarten": *Allgemeine Einstellungen*, *Einrichten von Klassen* und *Ausnahmeregelungen*, in denen die Parameter des Systems bearbeitet werden können.

Allgemeine Einstellungen

Hier können die Netzwerk-Devices und deren Bandbreiten, eine Monatsobergrenze für den verursachbaren Traffic sowie Pfade zu benötigten Programmen und zu einem Logfile angegeben werden. Weiterhin kann eingestellt werden, ob eine automatische Regelung der Bandbreiten stattfinden soll und ab wann Nutzer, die in eine schlechtere Klasse eingestuft wurden, per E-Mail benachrichtigt werden sollen:

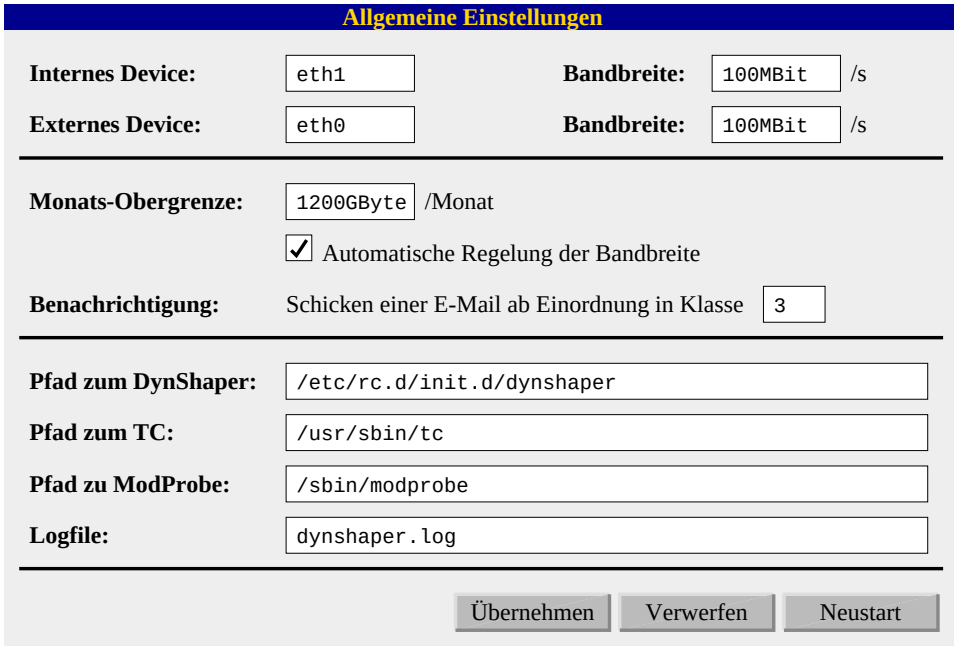


Abbildung 4.4-2.: Allgemeine Einstellungen

"Übernehmen" sichert die geänderten Einstellungen, "Verwerfen" setzt das Formular auf die Anfangswerte zurück. Durch Betätigen des Buttons "Neustart" kann der *DynShaper* nach erfolgten Änderungen mit den neuen Einstellungen gestartet werden.

Einrichten von Klassen

In dieser Registerkarte können die Parameter der Klassen (Rate, Richtung, IP-Adressen usw.) festgelegt werden:

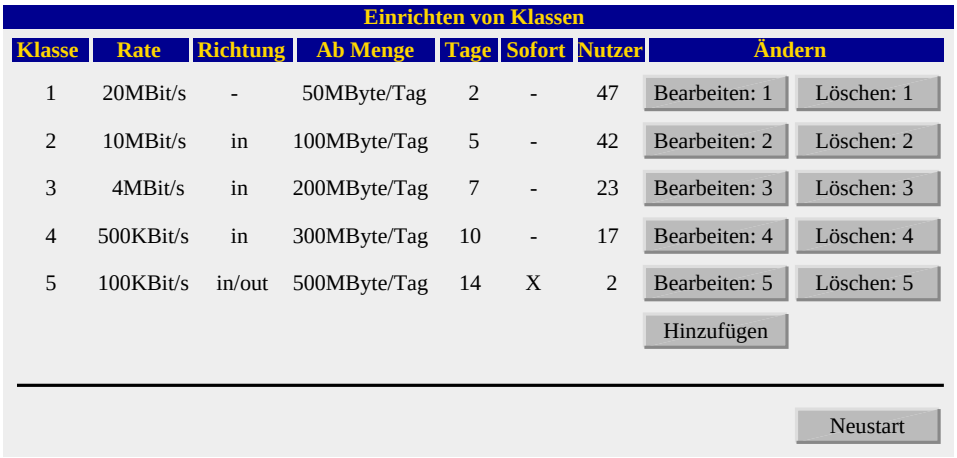


Abbildung 4.4-3.: Einrichten von Klassen

Durch "Hinzufügen" kann eine Klasse hinzugefügt und durch "Bearbeiten" und "Löschen" verändert bzw. entfernt werden. Die Klassen werden dabei immer nach ihrer Rate absteigend sortiert, da dies vom *DynShaper*-System vorausgesetzt wird.

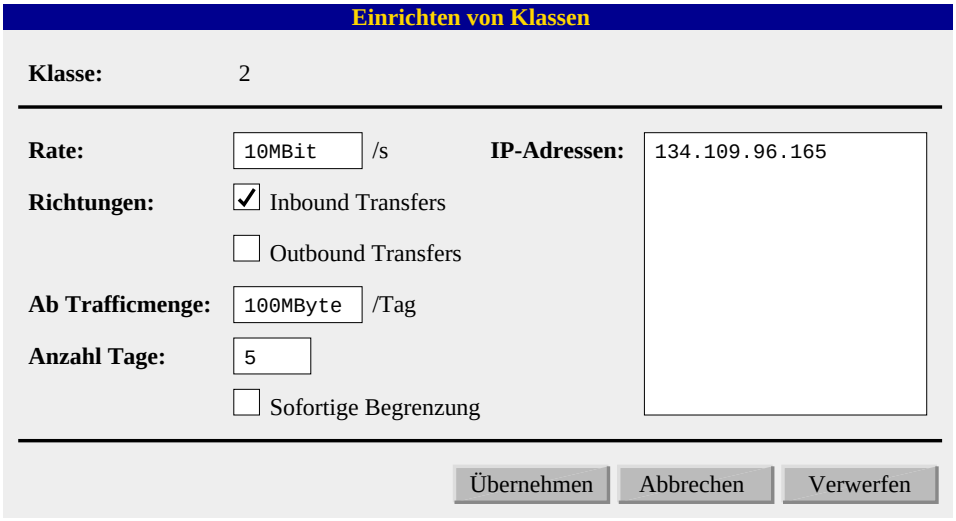


Abbildung 4.4-4.: Einrichten von Klassen - Eingabemaske

In der Eingabemaske für die Klassen können folgende Parameter geändert werden:

- "Rate" legt die Bandbreite fest, die möglichst kleiner als die Bandbreite der Netzinterfaces sein sollte; als Einheiten sind "KBit" und "MBit" zulässig, wird keine Einheit angegeben, wird "MBit" verwendet

- "Inbound/Outbound Transfers" legen die Richtungen fest, in die der Traffic begrenzt werden soll

- mit "Trafficmenge" und "Tage" kann man einstellen, ab wann und für wie lange man in diese Klasse eingeordnet wird

- "Sofortige Begrenzung" ist für Extremfälle sinnvoll, z.B. für die schlechteste Klasse

- die "IP-Adressen" der in diese Klasse eingeordneten Nutzer können auch bearbeitet werden, die Änderungen bleiben aber nur bis zum nächsten Update aus der Datenbank erhalten, die dortigen Datensätze werden hiervon nicht beeinflusst!

Der Button "Übernehmen" sichert die geänderten Einstellungen, "Abbrechen" verlässt die Eingabemaske und kehrt zur Übersicht der Klassen zurück. "Verwerfen" verhält sich analog zu den allgemeinen Einstellungen.

Ausnahmeregelungen

Hier können Ausnahmeregelungen, z.B. Sonderbehandlungen von bestimmten Protokollen, Nutzern oder Paketen, die zu bestimmten Adressen geschickt werden, definiert werden. Das ist u.a. sinnvoll, um Zugriffe auf lokale Ressourcen wie FTP-Server der Universität von der Bandbreitenbeschränkung auszuschließen:

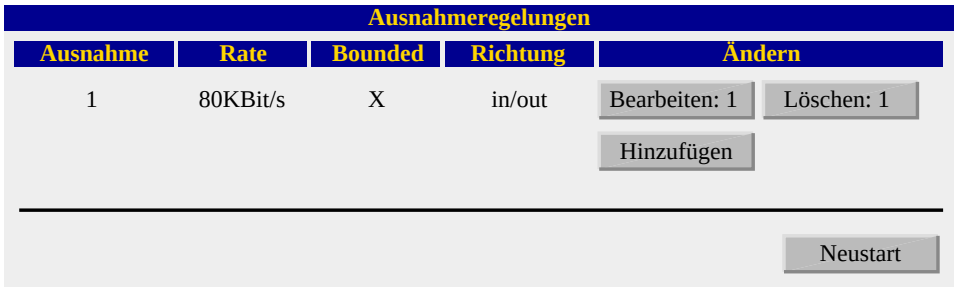


Abbildung 4.4-5.: Ausnahmeregelungen

Durch "Hinzufügen" kann eine Ausnahmeregelung hinzugefügt und durch "Bearbeiten" und "Löschen" analog zum Einrichten der Klassen verändert bzw. entfernt werden.

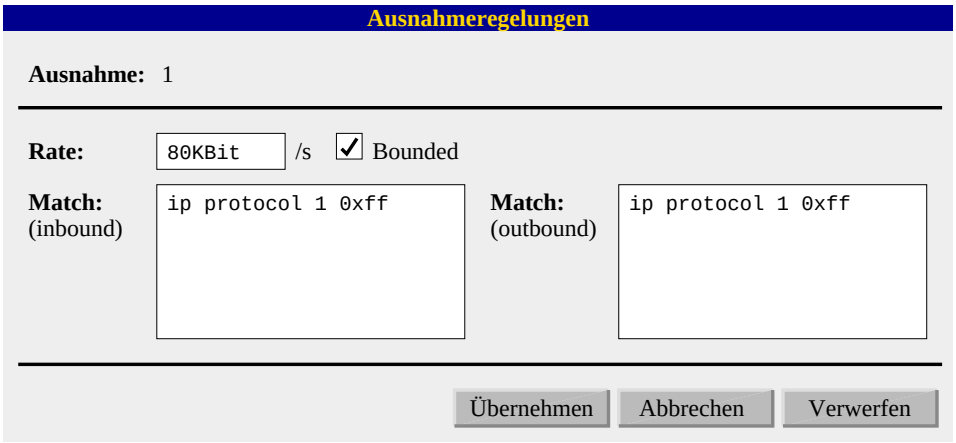


Abbildung 4.4-6.: Ausnahmeregelungen - Eingabemaske

In der zugehörigen Eingabemaske können die Parameter der Ausnahmeregelung geändert werden:

- "Rate" legt die Bandbreite fest; als Einheiten sind "KBit" und "MBit" zulässig, wird keine Einheit angegeben, wird wieder "MBit" verwendet
- mit "Bounded" kann eingestellt werden, ob die Bandbreite hart begrenzt werden soll; ist das der Fall, wird als QDisc für diese Ausnahme TBF verwendet, sonst SFQ
- die "Match"-Anweisungen legen fest, welche Pakete hier eingeordnet werden; zu beachten ist, dass ein Paket unter die betreffende Ausnahmeregelung fällt, wenn *mindestens eine* der Regeln zutrifft

Die Buttons "Übernehmen", "Abbrechen" und "Verwerfen" verhalten sich analog zu ihren Äquivalenten beim Einrichten der Klassen.

4.4.5. Installationsanleitung und Verzeichnisstruktur

Die einzelnen Komponenten müssen so installiert werden, dass der Webserver Zugriff auf jede besitzt. Folgende Verzeichnisstruktur wird empfohlen:

DynShaper Manager

```
htdocs/dynshaper/manager/dsmanager.php
htdocs/dynshaper/manager/dsfunctions.inc
htdocs/dynshaper/manager/xmlrpc.inc
```

DynShaper Watcher

```
htdocs/dynshaper/watcher/dswatcher.php
htdocs/dynshaper/watcher/dsfunctions.inc
htdocs/dynshaper/watcher/xmlrpc.inc
```

Der *Manager* und der *Watcher* können auf unterschiedlichen Rechnern laufen. Falls sie auf demselben Rechner im gleichen Verzeichnis installiert werden, muss sichergestellt werden, dass auf den *Manager* nur die Administratoren Zugriff besitzen (siehe nächster Abschnitt).

DynShaper Configurator

```
htdocs/dynshaper/configurator/dsconfigurator.php
htdocs/dynshaper/configurator/dsfunctions.inc
htdocs/dynshaper/configurator/xmlrpc.inc
htdocs/dynshaper/configurator/xmlrpcs.inc
/etc/rc.d/init.d/dynshaper
/etc/dynshaper.conf
```

DynShaper Evaluator

```
htdocs/dynshaper/evaluator/dsevaluator.php
htdocs/dynshaper/evaluator/dsevaluator.inc
htdocs/dynshaper/evaluator/dsfunctions.inc
htdocs/dynshaper/evaluator/xmlrpc.inc
htdocs/dynshaper/evaluator/xmlrpcs.inc
```

Auch der *Configurator* und der *Evaluator* können auf verschiedenen Rechnern installiert werden. Sollen sie auf demselben Rechner im gleichen Verzeichnis installiert werden, ist hier nichts weiter zu beachten.

Sollen alle vier Komponenten im selben Verzeichnis liegen, muss auf die Einstellung der Zugriffsbeschränkungen besondere Sorgfalt verwandt werden (siehe nächster Abschnitt).

In `dsfunctions.inc` sind nun noch die XML-RPC-URLs und der Pfad zur Konfigurationsdatei anzupassen. Letzteres ist ebenfalls im `dynshaper` vorzunehmen. Der Datenbankzugriff muss in `dsevaluator.inc` konfiguriert werden. Wenn der *Evaluator* als Cron-Job eingetragen ist, dann ist das System lauffähig.

4.4.6. Sicherheit des Systems

Authentifizierung der Nutzer und Administratoren

Um die Nutzer und Administratoren des *DynShapers* authentifizieren zu können, kann die Möglichkeit über das `.htaccess`-File genutzt werden:

```
SSLRequireSSL
AuthType Basic
AuthName DynShaper
AuthDBMFile /path/to/passwd.dbm
# AuthUserFile /path/to/passwd
Require valid-user
# Require user ...
```

Diese Methode sollte für die Administrations- und Abfrage-Skripte verwendet werden. Bei den XML-RPC-Servern funktioniert diese Variante nicht, da hier der Webserver als Nutzer auftritt und deshalb eine derartige Authentifizierung nicht stattfinden kann. Stattdessen sollte die Zugriffskontrolle rechnerbasiert erfolgen, indem nur der Zugriff des Computers, auf dem die erstgenannten Skripte liegen, erlaubt wird:

```
Order deny,allow
Deny from all
Allow from localhost  # wenn alles auf selbem Rechner
Allow from csn-server.csn.tu-chemnitz.de
```

Es muss aber sichergestellt werden, dass kein normaler Nutzer Zugang zu diesem Computer hat (abgesehen vom Webzugang), damit kein Unbefugter Zugriff auf die XML-RPC-Server erlangt.

Sind mehrere Komponenten, die unterschiedliche Zugriffsbeschränkungen besitzen, im selben Verzeichnis installiert, sollte mittels `<Files>` eine differenzierte Behandlung vorgenommen werden:

```
<Files dsconfigurator.php>
  Order deny,allow
  ...
</Files>
```

Vergabe von Root-Rechten

Da Queuing Disciplines, Klassen und Filter nur vom Nutzer `root` angelegt und verändert werden dürfen, es in diesem Fall aber auch vom Webserver aus möglich sein muss, unter dessen Nutzer-ID die Skripte laufen, müssen diesem über geeignete Mechanismen Root-Rechte verliehen werden. Zum Laden der Module sind diese ebenfalls notwendig.

Eine Variante wäre es, `dynshaper` vom *Configurator* aus über `sudo` mit Root-Rechten zu starten. Dazu ist folgender Eintrag in `/etc/sudoers` vorzunehmen:

```
httpd   ALL=(root) NOPASSWD: /etc/rc.d/init.d/dynshaper
```

Dieser Eintrag bewirkt, dass der Nutzer `httpd` ohne Passwortabfrage das angegebene Programm mit den Rechten von `root` starten darf. Über das Webinterface ist dafür noch der Pfad zum DynShaper um `/usr/bin/sudo` zu ergänzen.

Eine andere Möglichkeit ist es, die Rechte von `/usr/sbin/tc` so anzupassen, dass bei der Ausführung dieses Werkzeugs die Nutzer- und Gruppen-ID von `root` gesetzt werden:

```
chown root.root /usr/sbin/tc   # gehoert root (Standard)
chmod 755 /usr/sbin/tc         # ausfuehrbar fuer alle
chmod +s /usr/sbin/tc          # setuid root
```

Solange "normale" Nutzer keinen Zugriff auf das System haben, stellt das keinen allzu schwerwiegenden Sicherheitsbruch dar. Ansonsten könnte man die Rechte dahingehend modifizieren, dass nicht für alle Nutzer die Ausführung erlaubt wird, sondern nur für eine bestimmte Gruppe, in die der Webserver-Nutzer mit einzuordnen ist:

```
chown root.tcexec /usr/sbin/tc
usermod -G tcexec[,...] httpd  # bzw. anderer Nutzername
                               # alte Gruppen mit angeben!
chmod 750 /usr/sbin/tc         # ausfuehrbar fuer root+tcexec
chmod +s /usr/sbin/tc          # setuid root
```

Zum Laden der Module mittels `modprobe` werden ebenfalls Root-Rechte benötigt. Wurde `dynshaper` bereits als Startup-Skript gestartet, dann lief es zu diesem Zeitpunkt mit Root-Rechten und konnte die Module laden. Da diese seitdem in Benutzung waren, muss `modprobe` nicht mit `setuid root` versehen werden.

4.5. Mögliche Alternativen und Erweiterungen

Alternativ zu den in dieser Arbeit vorgeschlagenen Verfahrensweisen sind auch noch folgende Varianten denkbar:

- Statt wie derzeit lange in einer schlechten Klasse eingeordnet zu sein, könnten die Nutzer auch nur wenige Tage dort bleiben und dafür in den besser bewerteten Klassen entsprechend länger. Die Klassen sollten dann anders dimensioniert werden, da die Nutzerverteilung anders sein wird. Allerdings besteht hierbei die Gefahr, dass sich nach einem gewissen Zeitraum sehr viele Nutzer in den ersten Klassen ansammeln, wodurch auch mehr Filterregeln vom System verarbeitet werden müssten. Inwieweit dieses Verfahren sinnvoll ist, könnte bei einem Test im laufenden Betrieb oder zum Beispiel in einer Projektarbeit festgestellt werden.
- Weiterhin könnte man bessere von schlechteren Klassen borgen lassen, da sonst die schlechtere Klasse eine höhere Bandbreite pro Nutzer bekommen könnte, wenn in der besseren Klasse wesentlich mehr Nutzer eingeordnet sind.
- Zur Zeit stellt der *DynShaper Manager* nur die IP-Adressen der in die jeweilige Klasse eingeordneten Nutzer dar. Diese können auch bearbeitet werden, die Änderungen bleiben aber nur bis zum nächsten Update aus der Datenbank erhalten. Sinnvoller wäre es, wenn der *Manager* die Klassendaten weiterhin vom *Configurator* liest, die Nutzerdaten jedoch vom *Evaluator* und dann Nutzer statt IP-Adressen anzeigt. Beim Speichern würden aus den Nutzern deren IP-Adressen generiert und vom *Configurator* in die Konfigurationsdatei eingetragen werden. Gleichzeitig führt der *Evaluator* ein Update der Datenbank durch, so dass die Änderungen erhalten bleiben. Diese Funktion wäre beispielsweise für Ausnahmeregelungen nützlich. Zur Zeit kann diese Funktionalität aber über die bereits existierenden Ausnahmeregelungen realisiert werden.
- Alternativ könnten auch sämtliche Inhalte der Konfigurationsdatei in die Datenbank übernommen werden, so dass nur noch die dortigen Einstellungen zur Konfiguration und Auswertung herangezogen werden müssten. Die weiterhin notwendige Konfigurationsdatei für den *DynShaper* könnte dann komplett aus der Datenbank generiert werden und würde nur noch vom *DynShaper* selbst ausgewertet werden. Da dies eine umfangreichere Änderung darstellt, könnte sie ebenfalls im Rahmen einer Projektarbeit realisiert werden.
- Die sofortige Begrenzung der Extremfälle ist derzeit nur für die schlechteste Klasse sinnvoll, da der Traffic, der zur Einordnung in diese Klasse geführt hat, sonst nach jeder Stunde noch einmal gezählt wird, wodurch der Nutzer schrittweise in die schlechteste Klasse wandert, obwohl sein Traffic gar nicht so hoch war. Ein Ausweg wäre eine neue Spalte in der Datenbank-Tabelle, wo die Zuordnung Nutzer zu Klasse gespeichert ist: Statt die Klasse sofort in die bisherige Spalte zu schreiben, deren Wert bei jeder neuen Auswertung wieder herangezogen wird, sollte die neue Klasse in die neue Spalte geschrieben und erst zum Tageswechsel in die bisherige Spalte übertragen werden.

Zusätzlich wären noch folgende Erweiterungen möglich:

- Alle Nutzer, die noch nicht durch eine Filterregel erfasst wurden, könnten in eine unbegrenzte, hochpriorisierte Klasse eingeordnet werden, wobei als Queuing Discipline SFQ verwendet wird.
- Statt reiner Tageslimits, nach denen die Einordnung in die Klassen erfolgt, wäre eine Kombination aus Tages-, Wochen- und Monatslimits denkbar, beispielsweise 100 MByte/Tag, 500 MByte/Woche und 1200 MByte/Monat, um in Klasse 2 zu kommen. Damit könnten sich zukünftige Arbeiten beschäftigen.
- Die 2-Uhr-Grenze mit erneutem Zählbeginn könnte durch eine kontinuierliche Auswertung des Traffics innerhalb einer bestimmten Zeitspanne, z.B. in den letzten 24h oder auch 96h (vergleiche 96h-Tarif bei Energienetzen [Naum97]) ersetzt werden. Das ist aber wahrscheinlich nicht unbedingt nötig, da auch im jetzigen Modell die vorhergehenden Tage mit berücksichtigt werden.
- Die gesammelten Daten vom TC könnten ausgewertet werden, so dass Klassen und Ausnahmen mit dem meisten Traffic bei der automatischen Regelung der Bandbreiten auch am meisten heruntergeregelt werden. Zur Zeit werden nur Klassen bei der Regelung berücksichtigt und alle gleichermaßen geregelt.

5. Einsatz im CSN

5.1. Struktur des Chemnitzer Studentennetzes

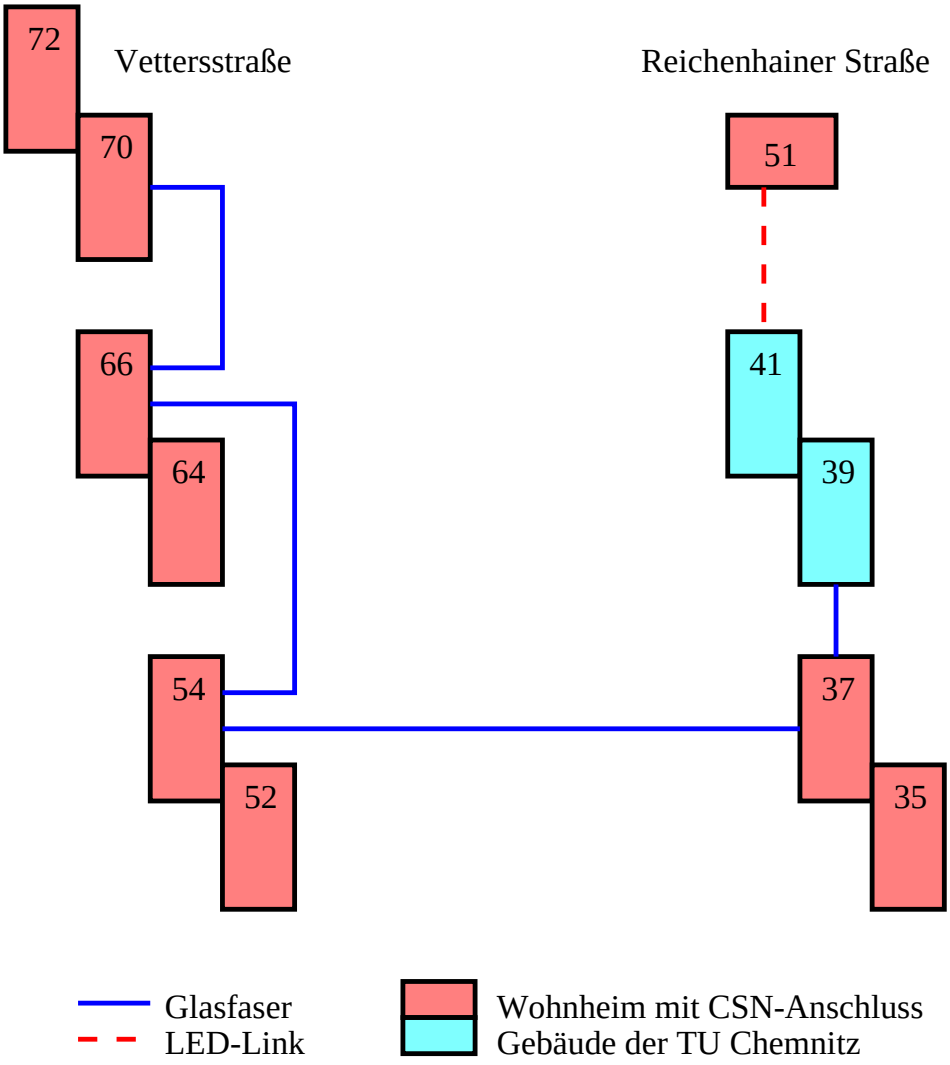


Abbildung 5.1-1.: Struktur des Chemnitzer Studentennetzes

Die Glasfaserstrecke zwischen der V54 und der Rh37 besteht aus acht Fasern, mit denen eine Vollduplex-Verbindung von 400 MBit/s möglich ist. An deren Enden existieren jeweils zwei gestapelte Switches 3com 3300FX, die als Backboneknoten fungieren und in denen die Subnetze als **Virtual Local Area Networks (VLAN)** definiert sind. Von der Rh37 aus erfolgt der Anschluss an das Campusnetz zur Rh39 mit 100 MBit/s. Die Rh51 ist über einen LED-Link angebunden.

Am Knoten in der Rh37 sind komplett mit 100 MBit/s alle Etagenknoten des Wohnblocks sowie zentrale Servertechnik und über den CSN-Server, auf dem die Bandbreitenbeschränkungen eingesetzt werden, der Uplink zum Rechenzentrum der TU angeschlossen. Dieser Knoten besitzt eine Hardware-Routingengine, die als zentraler Router des CSN fungiert.

Die Wohnheime der Vettersstraße hängen alle am Knoten in der V54. Die Etagen der V54 besitzen Uplinks mit 100 MBit/s und sind direkt an den zentralen Knoten angeschlossen. Die Uplinks aus der V52 mit 10 MBit/s werden auf einem Switch zusammengeführt, der dann per 100 MBit/s angeschlossen wird. Die Häuser V64/66 besitzen jeweils einen zentralen Switch und je einen Repeater für zwei Etagen. Uplink an den zentralen Knoten ist 100 MBit/s. In der V70/72 gibt es Uplinks mit 10 MBit/s, die per Medienwandler (Glasfaser auf Twisted Pair) angeschlossen sind.

5.2. Auswertung des Datenmaterials

Durch eine Datenbankabfrage der CSN-Datenbank standen die Trafficmengen für jeden Nutzer und Tag vom 20.12.2000 bis 13.04.2001 zur Verfügung. Bei der Auswertung wurden aber nur die komplett vorhandenen Monate Januar bis März 2001 betrachtet. Die Anzahl der innerhalb dieser drei Monate angeschlossenen Rechner betrug 1561. Nicht jeder dieser Rechner war seit Januar vorhanden, diese Tatsache wird aber bei der Abschätzung im nächsten Abschnitt berücksichtigt.

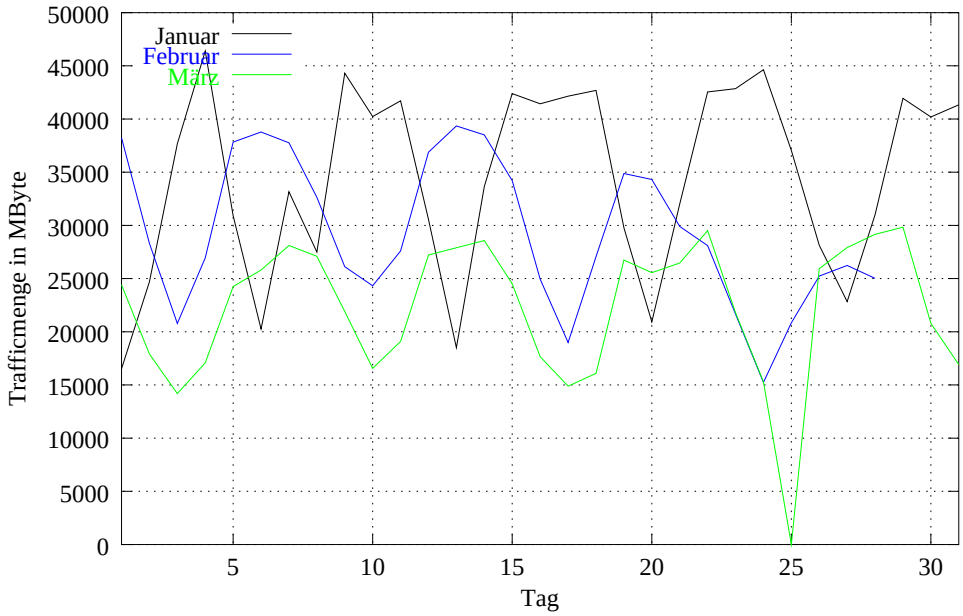


Abbildung 5.2-1.: Trafficmenge pro Tag in MByte

Die regelmäßigen Einbrüche sind auf das Wochenende und den damit geringeren Netzverkehr wegen kleinerer Nutzerzahlen zurückzuführen. Im März war wegen der Semesterferien generell eine geringere Trafficmenge zu verzeichnen. Der Wert 0 am 25. März kommt dadurch zustande, dass zur Zeit der Datenbankauswertung, die immer um 2 Uhr stattfindet, die Uhren auf Sommerzeit umgestellt wurden, weshalb an diesem Tag keine Eintragung vorgenommen wurde.

	Januar	**Februar**	**März**
Durchschnitt pro Tag:	34.517	29.311	22.973
Maximalwert im Monat:	46.387	39.353	29.840
Minimalwert im Monat:	16.527	15.233	14.193
Gesamtmenge:	1.070.032	820.722	689.189
Genormt auf 31 Tage:	1.070.032	908.656	712.162

Tabelle 5.2-1.: Trafficmenge pro Monat in MByte

In der nachfolgenden Tabelle sind die 10 größten Mengen, die pro Tag von einem Nutzer verursacht wurden, die 10 größten Mittel eines Nutzers über alle betrachteten Tage, die 10 Maxima, die an jeweils einem Tag pro Nutzer erreicht wurden, und die 10 größten Tagesdurchschnitte über alle Nutzer aufgelistet. Diese Extremfälle müssen durch die Bandbreitenbeschränkungen sofort erfasst werden (stündlich), um größeren Schaden zu verhindern.

Platz	Max. pro Nutzer	Mittel pro Nutzer	Max. pro Tag	Mittel pro Tag
1.	2303	228	2303	35,42
2.	2117	215	2117	34,56
3.	1953	188	1953	34,34
4.	1389	186	1389	34,18
5.	1315	172	1315	33,70
6.	1269	169	1269	33,66
7.	1156	161	1093	33,65
8.	1093	154	1058	33,65
9.	1058	153	1001	33,63
10.	1007	150	947	33,21

Tabelle 5.2-2.: Top Ten der Transfervolumina pro Nutzer und Tag in MByte

Es ist festzustellen, dass sich die Maxima pro Tag und pro Nutzer weitestgehend decken, erst ab dem 7. Platz gibt es Unterschiede. Das tritt dann ein, wenn an einem Tag mehrere solcher Extremfälle auftreten, aber bei den Tagesmaxima nur der größte davon erfasst wird, so dass dort in den Top Ten die nächsten Werte nachrücken. Weiterhin ist zu sehen, dass sich die Nutzer im Mittel an die 250MByte-Limits halten, und auch die Tagesdurchschnitte bewegen sich in einem vertretbaren Rahmen.

Um die Initialgrenzen für die Verkehrsklassen festlegen zu können, ist es wichtig herauszufinden, wie viele Nutzer ihre Limits bisher ausgeschöpft haben, und mit welcher Häufigkeit[1] das geschehen ist.

[1] Die Prozentangaben beziehen sich auf die Anzahl der Tage, in denen das betreffende Volumen überschritten wurde, im Verhältnis zur Anzahl der Tage, an denen überhaupt Netzverkehr stattfand, um Rechner, die erst gegen Ende des betrachteten Zeitraumes angeschlossen wurden, besser erfassen zu können.

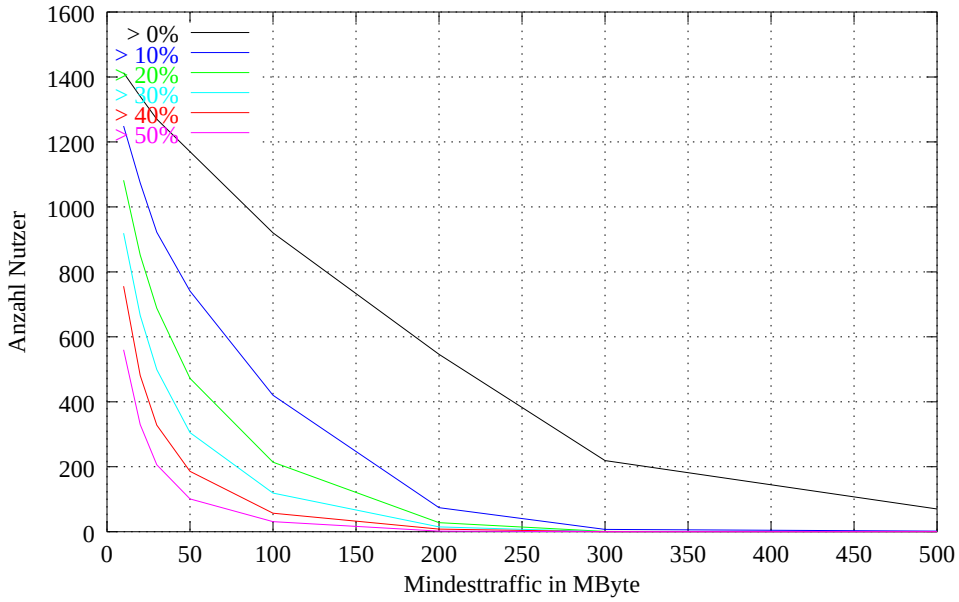


Abbildung 5.2-2.: Überschreitung bestimmter Transfervolumina

Transfervolumen	**>0%**	**>10%**	**>20%**	**>30%**	**>40%**	**>50%**
>10 MByte/Tag	1413	1249	1082	919	756	560
>20 MByte/Tag	1340	1074	852	666	481	330
>30 MByte/Tag	1270	922	688	499	328	206
>50 MByte/Tag	1170	741	472	305	186	101
>100 MByte/Tag	920	420	214	119	57	31
>200 MByte/Tag	546	74	28	15	8	2
>300 MByte/Tag	219	7	1	0	0	0
>500 MByte/Tag	70	2	0	0	0	0

Tabelle 5.2-3.: Überschreitung bestimmter Transfervolumina

Interessant ist hier vor allem die Grenze von 200 MByte/Tag, da bei diesen Nutzern voraussichtlich damit zu rechnen ist, dass sie am ehesten dazu neigen werden, viel Netzverkehr zu verursachen, wenn die Limits abgeschafft sind. Die dadurch stattfindende Erhöhung des Gesamttraffics soll im nächsten Abschnitt genauer untersucht werden.

5.3. Festlegung der Parameter für die Klassen

Aus dem vorigen Abschnitt sollen nun die ermittelten Werte herangezogen werden, um die Parameter (Bandbreite, Limits, Anzahl Tage) der Verkehrsklassen festzulegen. Diese Parameter sind dabei nur als Initialwerte zu verstehen, die konkreten Werte sollen durch die automatische Regelung der Bandbreiten ermittelt werden. Wichtig für die Initialwerte ist ein faires Verhältnis von Limits und Bandbreiten zwischen den Klassen, da dieses Verhältnis auch bei der Regelung immer bestehen bleibt.

Die Anzahl der Tage in einer Klasse sollte so festgelegt sein, dass sie in etwa der verursachten Menge geteilt durch 30 MByte (der Menge, die jedem Nutzer pro Tag theoretisch zustehen würde) entspricht, da dadurch der Traffic in diesen Tagen nach unten korrigiert werden kann, so dass am Ende der Zeitspanne nur Traffic von durchschnittlich 30 MByte/Tag (oder in derselben Größenordnung, da nicht davon auszugehen ist, dass dies alle Nutzer ausnutzen) erzeugt wurde. Die Bandbreiten der Klassen sollten immer unter der kleinsten Bandbreite der beteiligten Netzinterfaces liegen, um noch Spielraum für die Regelung zu lassen, da durch diese das Verhältnis der Bandbreiten untereinander nicht geändert wird und eine Überschreitung der tatsächlich verfügbaren Bandbreite nicht sonderlich sinnvoll ist.

Klasse	Ab Menge	Bandbreite	Tage	Begrenzung	Sofort
1	50 MByte/Tag	20 MBit/s	2	-	-
2	100 MByte/Tag	10 MBit/s	5	X	-
3	200 MByte/Tag	4 MBit/s	7	X	-
4	300 MByte/Tag	500 KBit/s	10	X	-
5	500 MByte/Tag	100 KBit/s	14	X	X

Tabelle 5.3-1.: Parameter der Klassen

Der verwendete Algorithmus legt bei mehrmaligen Überschreitungen der Klassenlimits die neue Klasse derart fest, dass die bisherige Klasse des Nutzers zu einem Anteil von 25% mit einfließt. Das bedeutet, wenn der Nutzer durch eine erste Übertretung eines Limits beispielsweise in Klasse 3 eingeordnet wird, würde er im Normalfall sieben Tage in dieser Klasse verweilen, danach noch fünf Tage in Klasse 2 und zwei Tage in Klasse 1, allerdings nur, solange er in dieser Zeit keine weiteren Limitüberschreitungen verursacht. Verletzt der Nutzer zum Beispiel nach zehn Tagen (also während er sich schon wieder in Klasse 2 befindet) die 200MByte-Grenze, wodurch er normalerweise erneut in Klasse 3 eingeordnet werden würde, fließt die Klasse 2 zu 25%, also 0,5 und aufgerundet 1, mit ein, weswegen der betreffende Nutzer letztendlich in Klasse 4 landen würde, wo er jetzt zehn Tage verweilt, dann sieben in Klasse 3 usw.

Dadurch handelt es sich um eine relativ faire Aufteilung, da z.B. ein Nutzer, der zweimal die 100MByte-Grenze überschreitet, genauso in Klasse 3 eingeordnet wird wie ein Nutzer, der einmalig die 200MByte-Grenze übertritt. Klasse 1 wird noch nicht begrenzt, sie dient nur der Speicherung der Information, welche Nutzer bereits die 50MByte-Grenze verletzt haben.

Extremfälle (>500 MByte/Tag) werden sofort begrenzt, damit sie keinen weiteren Schaden anrichten können.

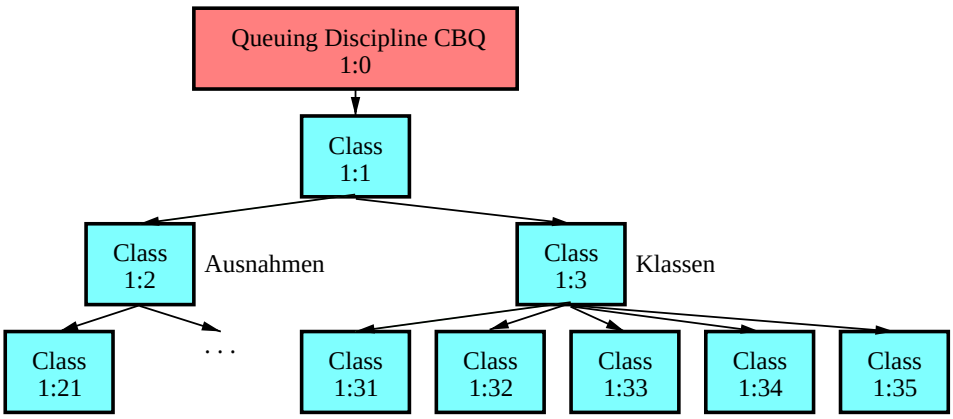


Abbildung 5.3-1.: Klassenbaum für ein Interface (Beispiel)

Um eine Abschätzung des erzeugbaren Traffics vornehmen zu können, werden die Messdaten anders aufgeschlüsselt:

Transfervolumen	**Nutzer mit >0%**	**Nutzer mit >10%**	**Nutzer mit >20%**
0-10 MByte/Tag	148	312	479
10-20 MByte/Tag	73	175	230
20-30 MByte/Tag	70	152	164
30-50 MByte/Tag	100	181	216
50-100 MByte/Tag	250	321	258
100-200 MByte/Tag	374	346	186
200-300 MByte/Tag	327	67	27
300-500 MByte/Tag	149	5	1
>500 MByte/Tag	70	2	0

Tabelle 5.3-2.: Einhaltung bestimmter Transfervolumina

Bei einer Abschaffung der alten 250MByte-Grenze ist nun damit zu rechnen, dass die Nutzer, die sich bisher am Rande des Limits bewegt haben (>200 MByte/Tag), ihren Traffic stark erhöhen. Es wird davon ausgegangen, dass die Hälfte der Nutzer, die bisher seltener als 10% mehr als 200 MByte/Tag verursacht haben, dies nun mit mehr als 20% tun werden.

Die folgende Tabelle zieht daher als Ausgangspunkt die Nutzer heran, die häufiger als 20% die festgelegten Limits überschritten haben, was die obige Annahme mit einschließt. Mit diesen Zahlen soll die Bandbreite pro Nutzer in der jeweiligen Klasse (unter der Annahme, dass 20% der Nutzer gleichzeitig das Netz nutzen) und der durch diese Klasse pro Tag erzeugbare

Traffic berechnet werden. Für die Nutzer, die keiner Klasse zugeteilt wurden, wird pauschal eine Trafficmenge von 20 MByte/Tag angenommen.

Klasse	Anzahl Nutzer	Bandbreite pro Nutzer	Erzeugbarer Traffic
1	258	1984 KBit/s	13 GByte/Tag
2	186	275 KBit/s	18 GByte/Tag
3	164	125 KBit/s	32 GByte/Tag
4	75	33 KBit/s	5 GByte/Tag
5	35	14 KBit/s	1 GByte/Tag
Rest	843	-	16 GByte/Tag

Tabelle 5.3-3.: Bandbreite pro Nutzer und erzeugbarer Traffic

Ab Klasse 4 sind die begrenzten Bandbreiten für die Obergrenze des erzeugbaren Traffics verantwortlich, darunter sorgt die Einhaltung der Klassenlimits dafür. Bei Klasse 1 wurden als zur Verfügung stehende Bandbreite 100MBit/s statt der eingestellten 20MBit/s herangezogen, da diese Klasse nicht beschränkt ist und daher die volle Bandbreite des Netzes nutzen kann.

In der Summe liegt der erzeugbare Traffic bei 85 GByte/Tag, was auf den Monat hochgerechnet die Vorgaben des CSN übersteigt. Allerdings handelt es sich hierbei um eine sehr pessimistische Abschätzung, da davon ausgegangen wurde, dass alle Nutzer, die häufiger als 20% diese Datenmengen verursachen, dies an jedem Tag gleichermaßen tun, obwohl das sicher nicht der Fall sein dürfte. Wenn der Traffic doch zu hoch sein sollte, greift die automatische Regelung der Bandbreiten ein und korrigiert diesen Wert entsprechend nach unten.

5.4. Durchgeführte Tests

5.4.1. Test 1: Benutzung der vorhandenen Daten

Der erste Test (der auf auf einem separaten Rechner erfolgte) verwendete die vorhandenen Daten, die auch schon zur Auswertung des Traffics benutzt wurden. Es wurden keinerlei Begrenzungsregeln aktiv, stattdessen sollte nur das Verhalten des Systems bei in Art und Anzahl realen Daten überprüft werden. Kritikpunkt an diesem Test ist, dass das System keine Auswirkungen auf den erzeugten Traffic hatte, wodurch er nicht mit Voraussagen über das veränderte Nutzerverhalten nach Abschaffung der Limits dienen kann. Weiterhin konnte nur mit den abschließenden Daten eines Tages gearbeitet werden, da die im Laufe des Tages entstandenen Werte nicht mehr vorlagen.

Für diesen Test wurde das Monatslimit auf 1000 GByte herabgesetzt, da der Traffic im Monat Januar nicht über 1200 GByte lag und die Bandbreitenregelung ansonsten nicht hätte getestet werden können. Es ergaben sich folgende Ergebnisse:

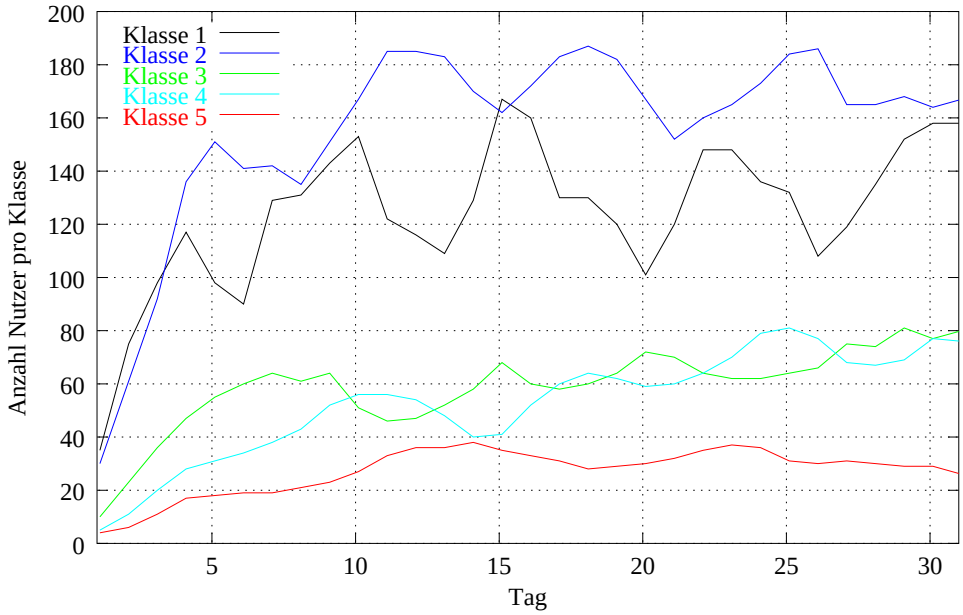


Abbildung 5.4-1.: Test 1: Anzahl der Nutzer in den Klassen

Tag	Klasse 1	Klasse 2	Klasse 3	Klasse 4	Klasse 5
01.01.	35	30	10	5	4
02.01.	75	61	23	11	6
03.01.	98	92	36	20	11
04.01.	117	136	47	28	17
05.01.	98	151	55	31	18
06.01.	90	141	60	34	19
07.01.	129	142	64	38	19
08.01.	131	135	61	43	21
09.01.	143	151	64	52	23
10.01.	153	167	51	56	27
11.01.	122	185	46	56	33
12.01.	116	185	47	54	36
13.01.	109	183	52	48	36
14.01.	129	170	58	40	38
15.01.	167	162	68	41	35
16.01.	160	172	60	52	33
17.01.	130	183	58	60	31
18.01.	130	187	60	64	28
19.01.	120	182	64	62	29
20.01.	101	167	72	59	30
21.01.	120	152	70	60	32
22.01.	148	160	64	64	35
23.01.	148	165	62	70	37
24.01.	136	173	62	79	36
25.01.	132	184	64	81	31
26.01.	108	186	66	77	30
27.01.	119	165	75	68	31
28.01.	135	165	74	67	30
29.01.	152	168	81	69	29
30.01.	158	164	77	77	29
31.01.	158	167	80	76	26

Tabelle 5.4-1.: Test 1: Anzahl der Nutzer in den Klassen

Die Anzahl der Nutzer bleibt immer in derselben Größenordnung. Die Werte für die Klassen 4 und 5 entsprechen den im vorigen Abschnitt vermuteten. Weiterhin ist zu entnehmen, dass mit ca. 350 Filterregeln gerechnet werden muss (Klasse 2 bis Klasse 5). Dieser Wert liegt in dem vermuteten Bereich und sollte vom Router ohne Probleme zu bewerkstelligen sein. Berechnet man analog zum vorigen Abschnitt die verfügbare Bandbreite pro Nutzer und den erzeugbaren Traffic, indem die Werte vom 31.01. herangezogen werden, erhält man folgende Tabelle:

Klasse	Anzahl Nutzer	Bandbreite pro Nutzer	Erzeugbarer Traffic
1	158	3241 KBit/s	8 GByte/Tag
2	167	307 KBit/s	16 GByte/Tag
3	80	256 KBit/s	16 GByte/Tag
4	76	33 KBit/s	5 GByte/Tag
5	26	19 KBit/s	1 GByte/Tag
Rest	1054	-	21 GByte/Tag

Tabelle 5.4-2.: Test 1: Bandbreite pro Nutzer und erzeugbarer Traffic

Es ergibt sich ein Gesamttraffic von 67 GByte/Tag. Dieser Wert ist zwar immer noch zu hoch, es gelten aber dieselben Einschränkungen wie oben angesprochen.

Die Bandbreiten der einzelnen Klassen entwickelten sich wie folgt:

Tag	Klasse 1	Klasse 2	Klasse 3	Klasse 4	Klasse 5
01.01.-05.01.	20,0 MBit/s	10,0 MBit/s	4,0 MBit/s	500 KBit/s	100 KBit/s
06.01.	21,2 MBit/s	10,6 MBit/s	4,2 MBit/s	530 KBit/s	106 KBit/s
07.01.	22,3 MBit/s	11,1 MBit/s	4,5 MBit/s	556 KBit/s	111 KBit/s
08.01.	23,5 MBit/s	11,7 MBit/s	4,7 MBit/s	587 KBit/s	118 KBit/s
09.01.-12.01.	24,2 MBit/s	12,1 MBit/s	4,8 MBit/s	604 KBit/s	121 KBit/s
13.01.-17.01.	24,6 MBit/s	12,3 MBit/s	4,9 MBit/s	616 KBit/s	123 KBit/s
18.01.	23,8 MBit/s	11,9 MBit/s	4,8 MBit/s	596 KBit/s	119 KBit/s
19.01.-22.01.	23,2 MBit/s	11,6 MBit/s	4,6 MBit/s	580 KBit/s	116 KBit/s
23.01.	22,5 MBit/s	11,3 MBit/s	4,5 MBit/s	563 KBit/s	113 KBit/s
24.01.	21,6 MBit/s	10,8 MBit/s	4,3 MBit/s	539 KBit/s	108 KBit/s
25.01.	20,6 MBit/s	10,3 MBit/s	4,1 MBit/s	515 KBit/s	103 KBit/s
26.01.	19,8 MBit/s	9,9 MBit/s	4,0 MBit/s	496 KBit/s	99 KBit/s
27.01.	19,3 MBit/s	9,7 MBit/s	3,9 MBit/s	483 KBit/s	97 KBit/s
28.01.	18,9 MBit/s	9,4 MBit/s	3,8 MBit/s	472 KBit/s	94 KBit/s
29.01.	18,3 MBit/s	9,1 MBit/s	3,7 MBit/s	457 KBit/s	91 KBit/s
30.01.	17,6 MBit/s	8,8 MBit/s	3,6 MBit/s	440 KBit/s	88 KBit/s
31.01.	16,9 MBit/s	8,4 MBit/s	3,4 MBit/s	421 KBit/s	84 KBit/s

Tabelle 5.4-3.: Test 1: Entwicklung der Bandbreiten der Klassen

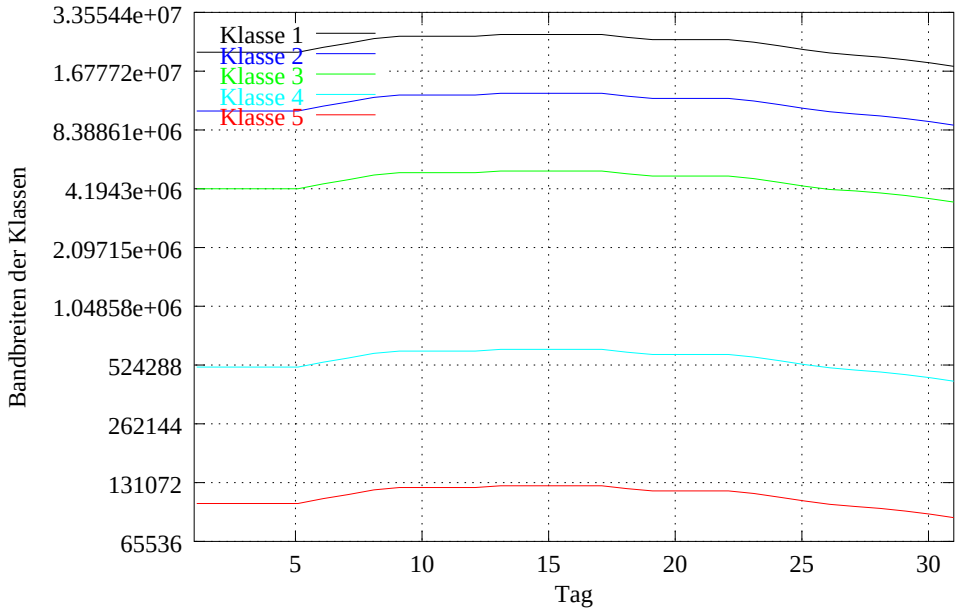


Abbildung 5.4-2.: Test 1: Entwicklung der Bandbreiten der Klassen

Zu Beginn liegt der hochgerechnete Monatstraffic noch unterhalb der 1000 GByte, weshalb die Bandbreiten erhöht werden. Ab der Mitte des Monats ist jedoch eine rückläufige Tendenz zu bemerken. Durch die Glättung der Anpassungsfaktoren (Werte nahe 1 werden auf 1 gesetzt) konnte das Maximum in der Monatsmitte abgeschwächt und dafür am Monatsende eine leicht höhere Bandbreite als ohne die Glättung erzielt werden.

Zusammengefasst kann man sagen, dass der erste Test zufriedenstellend verlief: Die Werte besitzen zwar keine große praktische Relevanz, da die Bandbreiten hier keinen Einfluss auf den Traffic hatten, aber sowohl die Anzahl der in Klassen eingeteilten Nutzer als auch die Bandbreiten der Klassen entsprachen den erwarteten Ergebnissen. Die Bearbeitungszeit von ca. 20s pro ausgewertetem Tag liegt auch in einem vertretbaren Rahmen.

5.4.2. Test 2: Test auf dem CSN-Server

Der zweite Test wurde auf dem CSN-Server durchgeführt, allerdings ohne aktive Begrenzung. Dadurch konnte das Verhalten des Systems unter realen Bedingungen überprüft und erfasst werden, welche Auswirkungen es auf den Durchsatz des Routers hat. Zusätzlich konnten auch schon erste Daten über die Nutzer gesammelt werden, damit die Klassen beim Einsatz bereits mit nahezu optimalen Werten versehen sind.

Die Korrektheit der SQL-Abfragen wurde vorher auf einer zur CSN-Datenbank identischen Datenbank überprüft, so dass von dieser Seite aus beim Einsatz nicht mehr mit Problemen zu rechnen war.

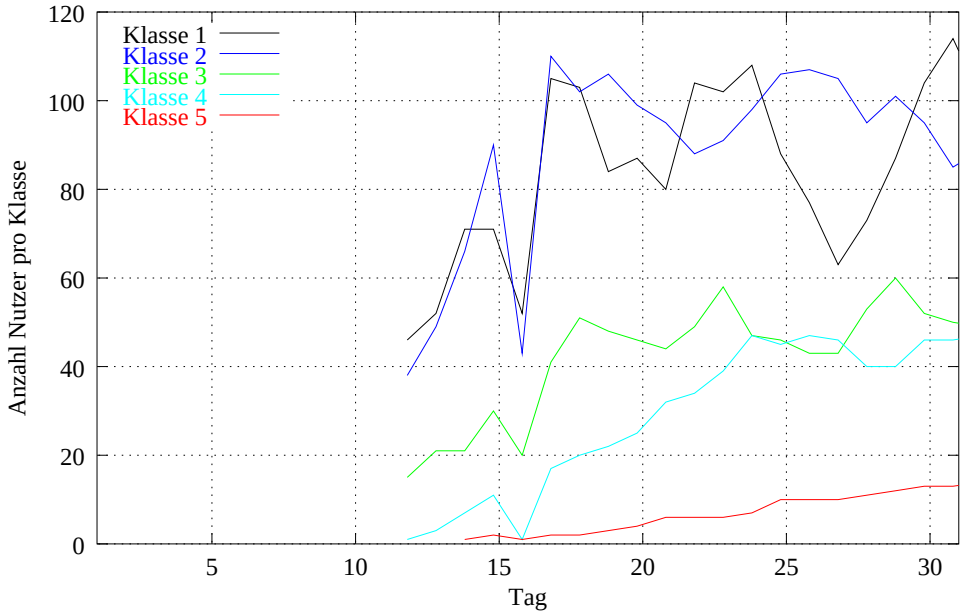


Abbildung 5.4-3.: Test 2: Anzahl der Nutzer in den Klassen

Die Werte liegen deutlich unter denen aus dem letzten Abschnitt. Dieser Umstand ist auf die Semesterferien und die Limits zurückzuführen, da die verbliebenen Nutzer nicht im gleichen Maße wie zur Vorlesungszeit Traffic verursachen konnten. Daher erübrigt sich auch eine Abschätzung der Bandbreiten pro Nutzer und des erzeugbaren Traffics, da diese Zahlen nicht repräsentativ für den späteren Einsatz sind.

Tag	Klasse 1	Klasse 2	Klasse 3	Klasse 4	Klasse 5
11.08.	46	38	15	1	0
12.08.	52	49	21	3	0
13.08.	71	66	21	7	1
14.08.	71	90	30	11	2
15.08.	52	43	20	1	1
16.08.	105	110	41	17	2
17.08.	103	102	51	20	2
18.08.	84	106	48	22	3
19.08.	87	99	46	25	4
20.08.	80	95	44	32	6
21.08.	104	88	49	34	6
22.08.	102	91	58	39	6
23.08.	108	98	47	47	7
24.08.	88	106	46	45	10
25.08.	77	107	43	47	10
26.08.	63	105	43	46	10
27.08.	73	95	53	40	11
28.08.	87	101	60	40	12
29.08.	104	95	52	46	13
30.08.	114	85	50	46	13
31.08.	100	89	49	47	14

Tabelle 5.4-4.: Test 2: Anzahl der Nutzer in den Klassen

Die fehlerhaften Werte am 15.08. resultieren aus fehlenden Daten, da an diesem Tag aufgrund eines Datenbank-Problems die bisherigen Klassen der Nutzer nicht ermittelt werden konnten.

Bei der Entwicklung der Bandbreiten der einzelnen Klassen gab es keine Überraschungen:

Tag	Klasse 1	Klasse 2	Klasse 3	Klasse 4	Klasse 5
01.08.-05.08.	20,0 MBit/s	10,0 MBit/s	4,0 MBit/s	500 KBit/s	100 KBit/s
06.08.	30,3 MBit/s	15,2 MBit/s	6,1 MBit/s	758 KBit/s	152 KBit/s
07.08.	46,0 MBit/s	23,0 MBit/s	9,2 MBit/s	1149 KBit/s	230 KBit/s
08.08.	69,8 MBit/s	34,9 MBit/s	13,9 MBit/s	1742 KBit/s	349 KBit/s
09.08.-31.08.	100,0 MBit/s	50,0 MBit/s	20,0 MBit/s	2497 KBit/s	500 KBit/s

Tabelle 5.4-5.: Test 2: Entwicklung der Bandbreiten der Klassen

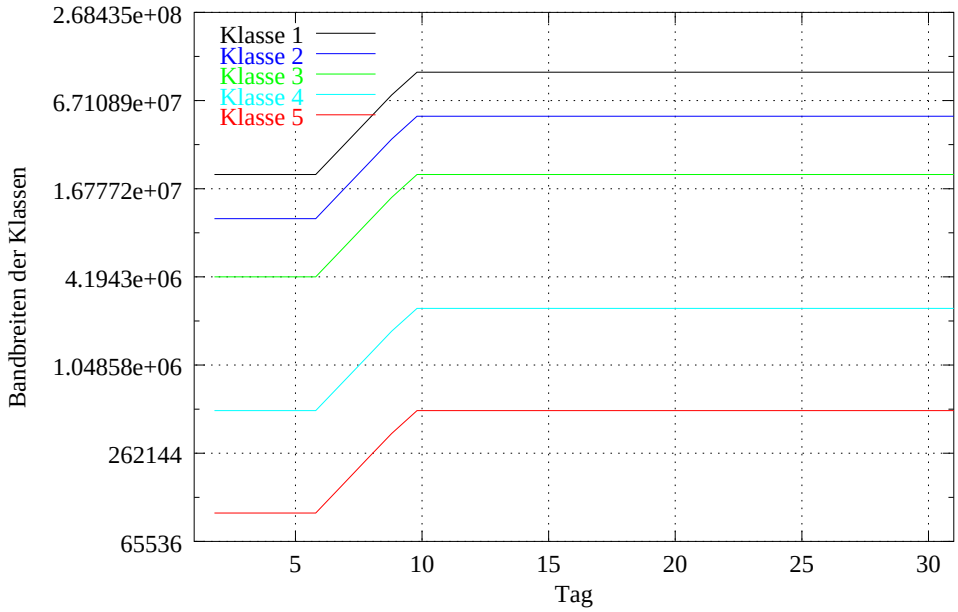


Abbildung 5.4-4.: Test 2: Entwicklung der Bandbreiten der Klassen

Da keine Beschränkungen aktiv waren, d.h. die Limits existierten noch und wurden meist auch eingehalten, und der Gesamttraffic des Monats aufgrund dieser Limits bisher immer unter der Monatsobergrenze gelegen hat, war damit zu rechnen, dass die Bandbreiten bis zum Maximum nach oben korrigiert werden. Da außerdem der Testzeitraum in die Semesterferien fiel, wurde diese Entwicklung noch beschleunigt.

Der Monatstraffic lag mit 572 GByte bei etwa der Hälfte des Traffics, der dem CSN zur Verfügung steht. Durch den Einsatz des *DynShapers* statt der Limits hätte diese zur Verfügung stehende Menge sicherlich besser ausgenutzt werden können.

6. Abschließende Bemerkungen

Zusammenfassung

Diese Arbeit beschäftigte sich mit der "schleichenden Abschaltung" begrenzter Netzzugänge mit Hilfe der Möglichkeiten, die der Linux-Kern hinsichtlich Quality of Service bietet. Nutzer, die bestimmte Datenvolumina überschreiten, werden schrittweise einer immer schlechter bewerteten Verkehrsklasse zugeordnet, in der ihnen weniger Bandbreite zur Verfügung steht und wo sie einige Tage verweilen, ehe sie schrittweise wieder in eine bessere Klasse eingeordnet werden.

Um das dem CSN zur Verfügung stehende Monatslimit möglichst optimal auszunutzen, ohne es jedoch zu überschreiten, erfolgt eine automatische Regelung der Bandbreiten der einzelnen Klassen.

Zur Administrierung des Systems wird eine webbasierte Benutzeroberfläche angeboten, mit der alle relevanten Parameter eingestellt werden können. Als Kommunikationsprotokoll kommt XML-RPC zum Einsatz. Die Implementation erfolgte hauptsächlich in PHP, das Skript für die eigentlichen Bandbreitenbegrenzungen ist jedoch ein einfaches Shellskript.

Seit August 2001 wird der *DynShaper* im Chemnitzer Studentennetz im Testbetrieb eingesetzt. Die Tests verliefen zufriedenstellend und lieferten in etwa die erwarteten Ergebnisse.

Ausblick

Ab dem 03.09.2001 läuft der *DynShaper* mit aktiven Begrenzungsregeln. Bisher sind noch keine nennenswerten Probleme aufgetreten. Zumindest für die Dauer dieses Monats bleiben die Limits allerdings noch erhalten, ab Oktober werden sie dann aber wahrscheinlich wegfallen. Als Vorteil erweist sich dabei die Tatsache, dass der Universität ab Oktober 2001 die doppelte Trafficmenge zur Verfügung steht. So dürfte es auch bei auftretenden Problemen nicht dazu kommen, dass die Monatsobergrenze überschritten werden kann.

Mögliche Alternativen und Erweiterungen, mit denen sich zukünftige Arbeiten beschäftigen könnten, sind in *4.5 Mögliche Alternativen und Erweiterungen* beschrieben.

A. Listings

A.1. Von Seite 35 (4.4.1): Vollständige Konfigurationsdatei

```
# This file was generated by DynShaper
# Modify with care!

DEVINT="eth1"
BWINT="100MBit"
WINT="10MBit"
DEVEXT="eth0"
BWEXT="100MBit"
WEXT="10MBit"
MLIMIT="1200GByte"
ADJUST="on"
SENDMAIL="3"
DYNSHAPER="/usr/bin/sudo /etc/rc.d/init.d/dynshaper"
TC="/usr/sbin/tc"
MODPROBE="/sbin/modprobe"
LOGFILE="dynshaper.log"

CLASSES="1 2 3 4 5"
RATES[1]="20MBit"
WEIGHTS[1]="2MBit"
INS[1]=""
OUTS[1]=""
LIMITS[1]="50MByte"
DAYS[1]="2"
ATONCE[1]=""
IPS[1]="134.109.96.101"
RATES[2]="10MBit"
WEIGHTS[2]="1MBit"
INS[2]="on"
OUTS[2]=""
LIMITS[2]="100MByte"
DAYS[2]="5"
ATONCE[2]=""
```

```
IPS[2]="134.109.96.165"
RATES[3]="4MBit"
WEIGHTS[3]="410KBit"
INS[3]="on"
OUTS[3]=""
LIMITS[3]="200MByte"
DAYS[3]="7"
ATONCE[3]=""
IPS[3]="134.109.116.159;134.109.116.85"
RATES[4]="500KBit"
WEIGHTS[4]="50KBit"
INS[4]="on"
OUTS[4]=""
LIMITS[4]="300MByte"
DAYS[4]="10"
ATONCE[4]=""
IPS[4]=""
RATES[5]="100KBit"
WEIGHTS[5]="10KBit"
INS[5]="on"
OUTS[5]="on"
LIMITS[5]="500MByte"
DAYS[5]="14"
ATONCE[5]="on"
IPS[5]=""

EXCEPTS="1"
ERATES[1]="80KBit"
EWEIGHTS[1]="8KBit"
EINS[1]="ip protocol 1 0xff"
EOUTS[1]="ip protocol 1 0xff"
EBOUNDS[1]="on"
```

A.2. Von Seite 36 (4.4.1): Festlegung der Ausnahmen

```
for except in ${EXCEPTS}; do
  if [ $device = ${DEVINT} -a "${EINS[$except]}" -o \
       $device = ${DEVEXT} -a "${EOUTS[$except]}" ]; then
    if [ ${EBOUNDS[$except]} ]; then bounded=" bounded"; else bounded=""; fi
    ${TC} class add dev $device parent 1:2 classid 1:2$except cbq \
          bandwidth $bandwidth rate ${ERATES[$except]} allot 1514 cell 8 \
          weight ${EWEIGHTS[$except]} prio 5 maxburst 20 avpkt 1000$bounded
    if [ $bounded ]; then
      ${TC} qdisc add dev $device parent 1:2$except tbf rate ${ERATES[$except]} \
            buffer 10Kb/8 limit 15Kb
    else
      ${TC} qdisc add dev $device parent 1:2$except sfq perturb 15 quantum 1514
    fi
    IFS=";"
    if [ $device = ${DEVINT} ]; then
      for match in ${EINS[$except]}; do
        cmd="${TC} filter add dev $device parent 1:0 protocol ip prio 100 u32 \
                   match $match flowid 1:2$except"
        eval $cmd  # noetig, da $match sonst gequotet
      done
    else
      for match in ${EOUTS[$except]}; do
        cmd="${TC} filter add dev $device parent 1:0 protocol ip prio 100 u32 \
                   match $match flowid 1:2$except"
        eval $cmd  # noetig, da $match sonst gequotet
      done
    fi
    IFS=$IFS2
  fi
done
```

A.3. Von Seite 36 (4.4.1): Definition der Klassen

```
for class in ${CLASSES}; do
  if [ $device = ${DEVINT} -a "${INS[$class]}" -o \
       $device = ${DEVEXT} -a "${OUTS[$class]}" ]; then
    ${TC} class add dev $device parent 1:3 classid 1:3$class cbq \
          bandwidth $bandwidth rate ${RATES[$class]} allot 1514 cell 8 \
          weight ${WEIGHTS[$class]} prio 5 maxburst 20 avpkt 1000 bounded
    ${TC} qdisc add dev $device parent 1:3$class tbf rate ${RATES[$class]} \
          buffer 10Kb/8 limit 15Kb
    if [ "${IPS[$class]}" ]; then
      IFS=";"
      if [ $device = ${DEVINT} ]; then
        for ip in ${IPS[$class]}; do
          ${TC} filter add dev $device parent 1:0 protocol ip prio 100 u32 \
                match ip dst $ip/32 flowid 1:3$class
        done
      else
        for ip in ${IPS[$class]}; do
          ${TC} filter add dev $device parent 1:0 protocol ip prio 100 u32 \
                match ip src $ip/32 flowid 1:3$class
        done
      fi
      IFS=$IFS2
    fi
  fi
done
```

Literaturverzeichnis

[AHSK99] Werner Almesberger, Jamal Hadi Salim, Alexey Kuznetsov: "Differentiated Services on Linux", 1999
ftp://lrcftp.epfl.ch/pub/linux/diffserv/misc/dsid-01.txt

[CSEZ93] Ron Cocchi, Scott Shenker, Debora Estrin, Lixia Zhang: "Pricing in Computer Networks: Motivation, Formulation, and Example", 1993
ftp://parcftp.xerox.com/pub/net-research/pricing2.ps.Z

[FeHu98] Paul Ferguson, Geoff Huston: "Quality of Service - Delivering QoS on the Internet and in Corporate Networks", Wiley Computer Publishing, 1998

[HeHo01] Sven Heese, Kai Hoelzner (verantw.): "WiN - G-WiN - Dienste - Internet", DFN, Stand 29.06.2001
http://www.dfn.de/win/gwin/dienste/internet.html

[Horb00a] Jan Horbach: "Studienarbeit - QoS and/or fair Queuing", TU Chemnitz, Professur Rechnernetze und verteilte Systeme, 2000

[Horb00b] Jan Horbach: "XML-RPC - RPC mit HTTP/XML", TU Chemnitz, Fakultät für Informatik, 2000
http://archiv.tu-chemnitz.de/pub/2000/0033/

[Lamb99] Mark Lamb: "iproute2+tc notes", 1999
http://snafu.freedom.org/linux2.2/iproute-notes.html

[Lore00] Mario Lorenz: "Geordnete Wege - Traffic Control mit Linux", iX 4/2000, S. 134-137, Verlag Heinz Heise GmbH &Co KG

[MHR91] Cyndi Mills, Donald Hirsh, Gregory Ruth: RFC 1272 "Internet Accounting: Background", 1991
http://www.ietf.org/rfc/rfc1272.txt

[Naum97] Torsten Naumann: "Diplomarbeit - Ressourcensteuerung für Internet-Zugänge", TU Chemnitz, Professur Rechnernetze und verteilte Systeme, 1997

[PHPG01] PHP Group: "PHP Hypertext Preprocessor", 2001
http://www.php.net

[PSQL01] "PostgreSQL", PostgreSQL Global Development Group, 2001
http://www.de.postgresql.org

[Radh99] Saravanan Radhakrishan: "Linux - Advanced Networking Overview", 1999
http://qos.ittc.ukans.edu/howto/

[Shen94] Scott Shenker: "Service Models and Pricing Policies for an Integrated Services Internet", Palo Alto Research Center, Xerox Corporation, 1994
ftp://parcftp.xerox.com/pub/net-research/policy.ps.Z

[Stev94] W. Richard Stevens: "TCP/IP Illustrated, Volume 1: The Protocols", Addison-Wesley Publishing Company, 1994

[Stro98] Dr. Günther Strohrmann: "Automatisierungstechnik 1", 4., überarbeitete und erweiterte Auflage, R. Oldenbourg Verlag München Wien, 1998

[ToBe89] Prof. Dr. sc. techn. Heinz Töpfer, Doz. Dr.-Ing. Peter Besch: "Grundlagen der Automatisierungstechnik", 2., durchgesehene Auflage, VEB Verlag Technik Berlin, 1989

[URZ01] URZ Autorenkollektiv: "Jahresrückblick 1999/2000 (Mitteilungen des URZ)", Archiv TU Chemnitz, 2001
http://archiv.tu-chemnitz.de/pub/2001/0014/data/jahr99-00.html

[Usef00] "XML-RPC Resources", Useful Information Company, 1999-2001
http://xmlrpc.usefulinc.com

[User01] UserLand: "XML-RPC Home Page", UserLand Software, Inc., 2001
http://www.xmlrpc.com

Selbständigkeitserklärung

Ich versichere hiermit eidesstattlich durch meine Unterschrift, dass ich die vorliegende Arbeit selbständig und ohne unzulässige fremde Hilfe angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Chemnitz, den 27. September 2001

Index