FACHHOCHSCHULE FURTWANGEN · SS 2001
COMPUTER NETWORKING
NETZPRAKTIKUM B

ADRIAN WOIZIK
MARCUS BENEDIX

# IMPLEMENTIERUNG VON QUALITY OF SERVICE (QOS) IM NETZWERK DES STUDENTEN-WOHNHEIMES GHB DER FH FURTWANGEN

Diese Dokument beschreibt die Implementierung von „Quality of Service (QoS)“ im Netzwerk des Studentenwohnheimes „Am Grosshausberg“ der FH-Furtwangen und ist die Dokumentation des Projektes im Rahmen der Wahlpflichtvorlesung „Netzwerkpraktikum B“ des Studienganges „Computer Networking (CN)“.
Dieses Dokument ist sowohl Ausarbeitung zum Projekt als auch Dokumentation für die Administratoren des GHB.

Furtwangen, im Juni 2001
Adrian Woizik
Marcus Benedix

# Inhaltsverzeichnis

**1 Einführung** **4**
1.1 Ausgangssituation . . . . . 4
1.2 Messungen . . . . . 6
1.3 Zielsetzung und Anforderungen . . . . . 7

**2 Theoretische Überlegungen** **8**
2.1 Bandbreiten Management unter dem Betriebssystem Linux . . . . . 8
2.2 Konzeption . . . . . 10
2.2.1 Konzept 1 . . . . . 10
2.2.2 Konzept 2 . . . . . 11
2.2.3 Konzept 3 . . . . . 11
2.3 Technische Einschränkungen . . . . . 12
2.4 Einsatzfähige Lösung . . . . . 13

**3 Umsetzung** **14**
3.1 Konzept . . . . . 14
3.2 Kernel . . . . . 14
3.3 Init-Scripte . . . . . 16
3.4 ipchains . . . . . 16

**4 Fazit** **17**

**A Konfigurationsfiles** **19**
A.1 Kernelconfig Linux-2.4.6 . . . . . 19
A.2 /usr/local/sbin/qos . . . . . 20
A.3 /usr/local/etc/qosfwchain . . . . . 24
A.4 /usr/local/etc/header/qosausnahmen . . . . . 26

# 1 Einführung

Das Studentenwohnheim „Am Grosshausberg (GHB)“ der FH Furtwangen beherbergt in 300 Appartements und 50 Wohnungen etwa 400 Studenten. Jede Wohneinheit verfügt über eine Anbindung an das interne Netz, welches wiederrum über einen 10 MBit-Laser bzw. über eine 2MBit-Richtfunkstrecke an das Netz der FH angebunden ist. Abbildung 1.1 stellt die Konfiguration des Netzwerkes grafisch dar.

## 1.1 Ausgangssituation

Derzeitig wird die Bandbreite des ein- und ausgehenden Traffics des Wohnheimes im Rechenzentrum der FH Furtwangen in der Zeit von 08:00 bis 22:00 Uhr auf insgesamt 1 MBit/s geshapt. Diese verfügbare Bandbreite müssen sich die 400 Nutzer der Wohnheime teilen. Hinzu kommt ein weiteres (nicht zeitabhängiges) Shaping in Freiburg für die Netze der Studentenwohnheime.
Die Wohnheime untereinander werden nicht über die Cisco im Rechenzentrum der FH gerouted und sind somit nicht von dem Shaping betroffen.

Zur effizienten Nutzung der möglichen Bandbreite steht den Anwendern im Wohnheim ein Proxy (deepSpace.ghb.fh-furtwangen.de) für HTTP und FTP zur Verfügung. Dieser Server ist in Freiburg vom Shaping ausgenommen. Außerdem helfen offizielle und inoffizielle FTP-Server bei der Vermeidung unnötigen Traffics durch mehrfachen Download derselben Dateien (z.B. Servicepacks für Windows).
Die derzeitige Konfiguration des GHB-Netzes und seines Gateways zum Rechenzentrum sieht keine Priorisierung des Trafficaufkommens anhand von Verkehrsarten vor.
Dies hat in Spitzenzeiten zur Folge, dass trafficintensive Anwendungen wie HTTP/FTP oder Audiostreaming den Großteil der zur Verfügung stehenden Bandbreite beanspruchen. Andere interaktive Anwendungen wie SSH werden hinsichtlich der verfügbaren Bandbreite durch effizientere Protokolle derart ausgebremst werden, daß ein Arbeiten in diesen Spitzenzeiten durch 1-2 Secunden RTT[1] fast nicht mehr möglich ist.
Praktisch kann im ungünstigsten Fall ein einzelner Anwender das Netz derart belasten, daß für andere Nutzer keine Bandbreite mehr übrig bleibt, damit sinnvoll gearbeitet wer-

[1] ***R**ound **T**rip **T**ime* - Übertragungszeit eines Paketes vom Sender zum Empfänger und zurück.

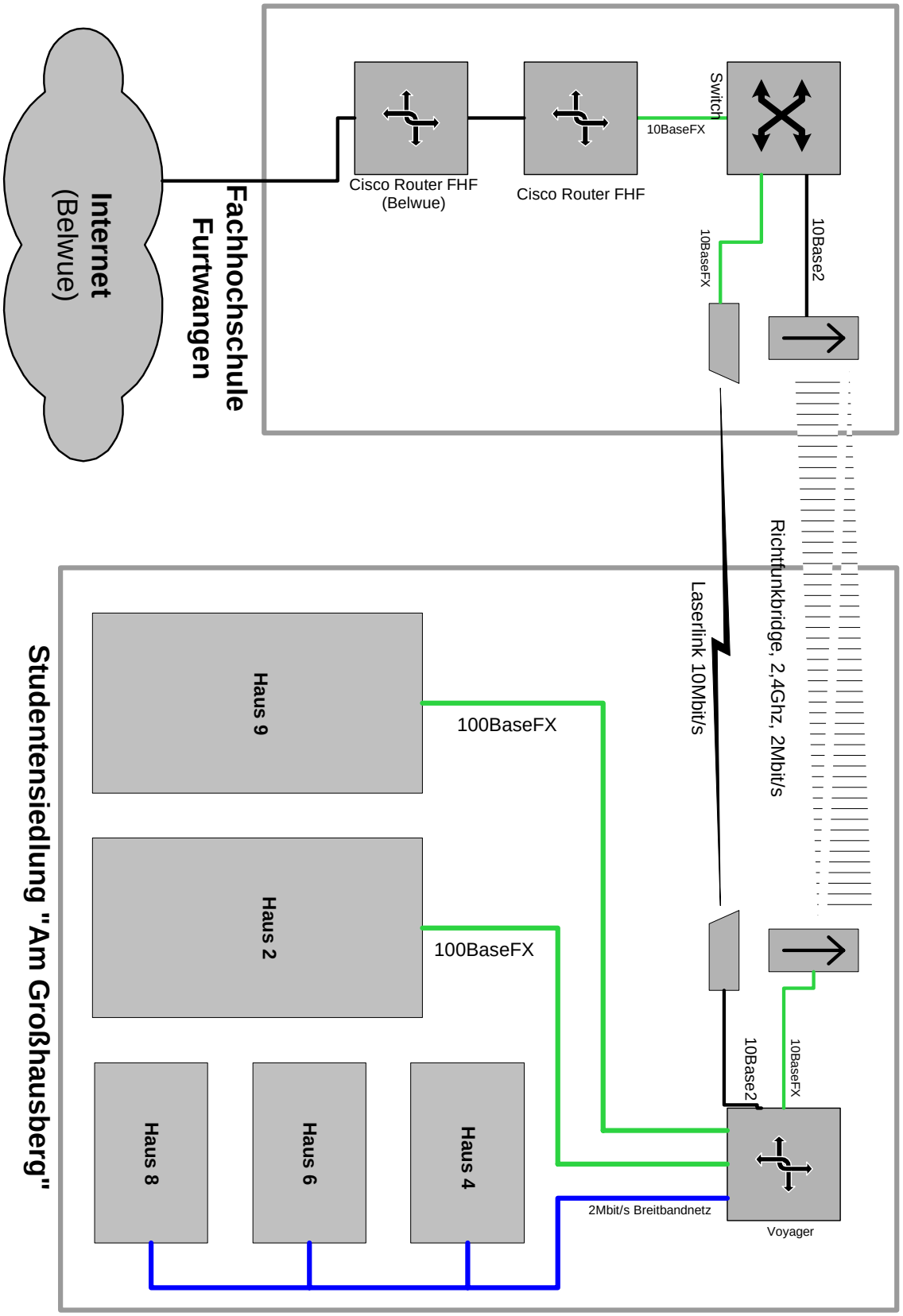


Abbildung 1.1: Netzstruktur der Wohnheime des GHB

den kann.
Verschärft wird diese Problematik, wenn infolge von Nebel die Anbindung an das Rechenzentrum der FH Furtwangen über den 10-MBit/s-Laser nicht mehr möglich ist und auf die Backup-Lösung über die 2-MBit/s-Richtfunkstrecke umgeschaltet werden muss.

## 1.2 Messungen

Die vermutete Problematik konnte durch Messungen unter Verwendung von tcpdump, tcptrace und xplot bestätigt werden. Die Abbildung 1.2 zeigt die durchschnittlichen Roundtrip-Zeiten einer SSH-Verbindung aus dem GHB zu einem Server im Foo-Pool des Studiengangs CN der FH Furtwangen ohne Einsatz von QoS im Verlauf zwischen 16:00 und 23:00 Uhr.

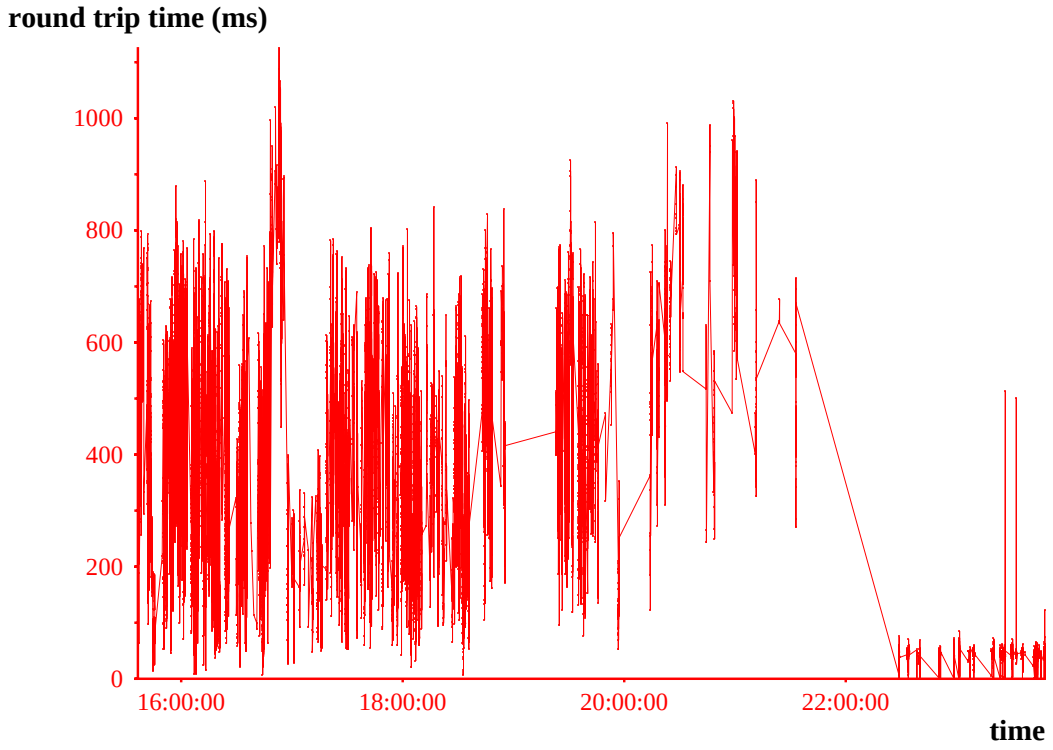


Abbildung 1.2: Paketlaufzeiten (Round-Trip-Time) einer ssh-Verbindung von realworld.ghb.fh-furtwangen.de nach father.foo.fh-furtwangen.de am 17. Juni 2001 zwischen 16:00 und 23:00 Uhr ohne Einsatz von QoS

Deutlich zu erkennen sind die recht hohen Paketlaufzeiten von durchschnittlich 600 ms in der Zeit von 16:00 und 22:00 Uhr. Ursache ist das Shaping im RZ der FH Furtwan-

| Einstufung | Priorität | Service | Beschreibung |
|---|---|---|---|
| sehr wichtig | 1 | ssh | Remote Access, Filetransfer |
| | 1 | icmp | Diagnose |
| | 2 | email, irc, news | Kommunikation |
| wichtig | 4 | http | „Surfen“ |
| vernachlässigbar | 5 | diverse Services | Traffic in andere Netze der FH |
| unwichtig | 7 | sonstige Services (ftp, audiostreaming) | alle anderen, nicht gesondert aufgeführten Services |

Tabelle 1.1: Spezifikation der Anforderungen für die Einführung von QoS im Netzwerk des Studentenwohnheims GHB

gen. Der hohe Anteil von anderem Traffic aus dem GHB führt zur Benachteiligung des SSH-Traffics. Sinnvolles Arbeiten mit ssh ist mit diesen Werten kaum möglich.
Nach 22:00 Uhr wird im RZ das Shaping der Wohnheimnetze abgeschalten. Dies wird durch den steilen Abfall der Round-Trip-Times auf durchschnittlich 100 ms deutlich.

## 1.3 Zielsetzung und Anforderungen

Zielsetzung des Projektes ist es, „Quality of Service (QoS)“ auf dem Gateway des Wohnheimes einzuführen und somit eine protokollabhängige Priorisierung und Bandbreitenbegrenzung zu implementieren. Rahmenbedingung für das Projekt ist das Betriebssystem Linux auf dem Router „wurmloch.ghb.fh-furtwangen.de“ des Wohnheimsnetzes mit seiner bestehenden Konfiguration. Da eine Vielzahl weiterer Anwendungen (Accounting, Firewall, ...) auf diese Konfiguration aufbauen, muss die zu entwickelnde Lösung auf diesem System basieren.
Desweiteren steht keine Testumgebung zur Verfügung. Die Tests und die Implementierung können damit nur auf einem Produktivsystem erfolgen. Eine lediglich geringfügige bis gar keine Beeinträchtigung der Nutzer im Wohnheim muss unbedingt angestrebt werden.
Nach Analyse des durch die Nutzer des Netzes verursachten Traffics, der bestehenden Probleme zu Hochlastzeiten und der Bedürfnisse konnten die Services klassifiziert werden:

# 2 Theoretische Überlegungen

## 2.1 Bandbreiten Management unter dem Betriebssystem Linux

Zur Realisierung eines Bandbreiten Managements am Beispiel QoS bietet Linux drei grundlegende Ansatzpunkte:

1. Queueing-Strategien (im englischen Queueing Disciplines)
2. (Traffic) Klassen
3. Filter

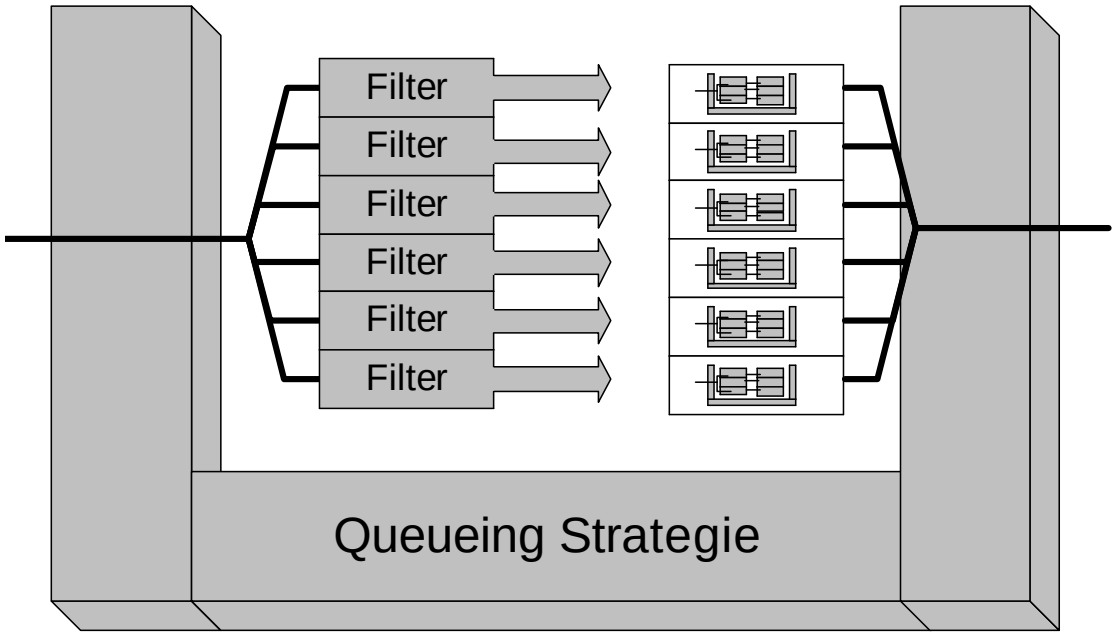


Abbildung 2.1: vereinfachte Darstellung der Blockstruktur des Linux Traffic Controls

Einer jeden Klasse wird eine Queueing Strategie zugeordnet. Diese Queueing Strategie kann eigene Klassen enthalten. Jede Queueing Strategie besitzt eine oder mehr Queues. Hier sehen wir schon die Flexibilität der Queueing Mechanismen bei Linux. Es können

nicht nur heterogene Klassen-Queueing-Strukturen, sondern auch rekursive Strukturen gebildet werden. Abbildung 2.1 zeigt die Blockstruktur des Linux Kernel Traffic Controls. Die Filter dienen dazu das die Pakete einer bestimmten Traffic Klasse zugeordnet werden können. Innerhalb dieser Traffic Klasse besteht die Möglichkeit weitere Queueing Strategien/Filter rekursive zu definieren. Am Ende dieser Kette stehen die Queueing Strategien, die über die Priorität, Queue-Größe, maximaler Bandbreite, usw entscheiden. Es sind dabei folgende Queueing Strategien wählbar:

1. Class Based Queueing (CBQ)
2. Token Bucket Flow (dasselbe wie Token Bucket Filter ) (TBF)
3. Clark-Shenker-Zhang (CSZ)
4. First In First Out (FIFO)
5. Priority
6. Traffic Equalizer (TEQL)
7. Stochastic Fair Queueing (SFQ)
8. Asynchronous Transfer Mode (ATM)
9. Random Early Detection (RED)
10. Generalized RED (GRED)
11. Diff-Serv Marker (DS_MARK)

Dank der rekursiven Möglichkeiten kann man als oberste Klasse (die als *root-class* bezeichnet wird) auf Class Based Queueing und in den unteren Ebenen auf SFQ zurückgreifen. Nicht jede Queueing Strategie ist für alles einsetzbar. *SFQ* ist eine simplifizierte Weight Fairness Queueing Methode. Der Vorteil an SFQ ist das es wenig CPU Last benötigt. Es handelt seine Tockens nach einem Round-Robin Verfahren ab, und ist deswegen nicht wirklich „Fair“, wie es der Name suggerieren mag. Doch für den Einsatz im GHB genügt das allemal als unterste Queueing Disziplin. Die Abhandlung nach Priorität, Bandbreite wird in den oberen Klassen von CBQ vorgenommen.
*CBQ* als Basis jeglichem *QoS* übernimmt die Hauptarbeit bei der Priorisierung der Pakete in ihre Klassen. Mit *CBQ* werden die Prioritäten und Bandbreiten beachtet. CBQ ermittelt anhand der Parameter der Klasse die Verögerung mit der ein Packet weitergeleitet wird. Dabei hat jede Klasse seine eigene Queue die ihre eigenen Parameter hat. U.a. auch die der Eltern-Klasse.

## 2.2 Konzeption

Am Router *Wurmloch.GHB.fh-furtwangen.de* ist wie in 2.2 abgebildet die Routing Struktur gegeben.

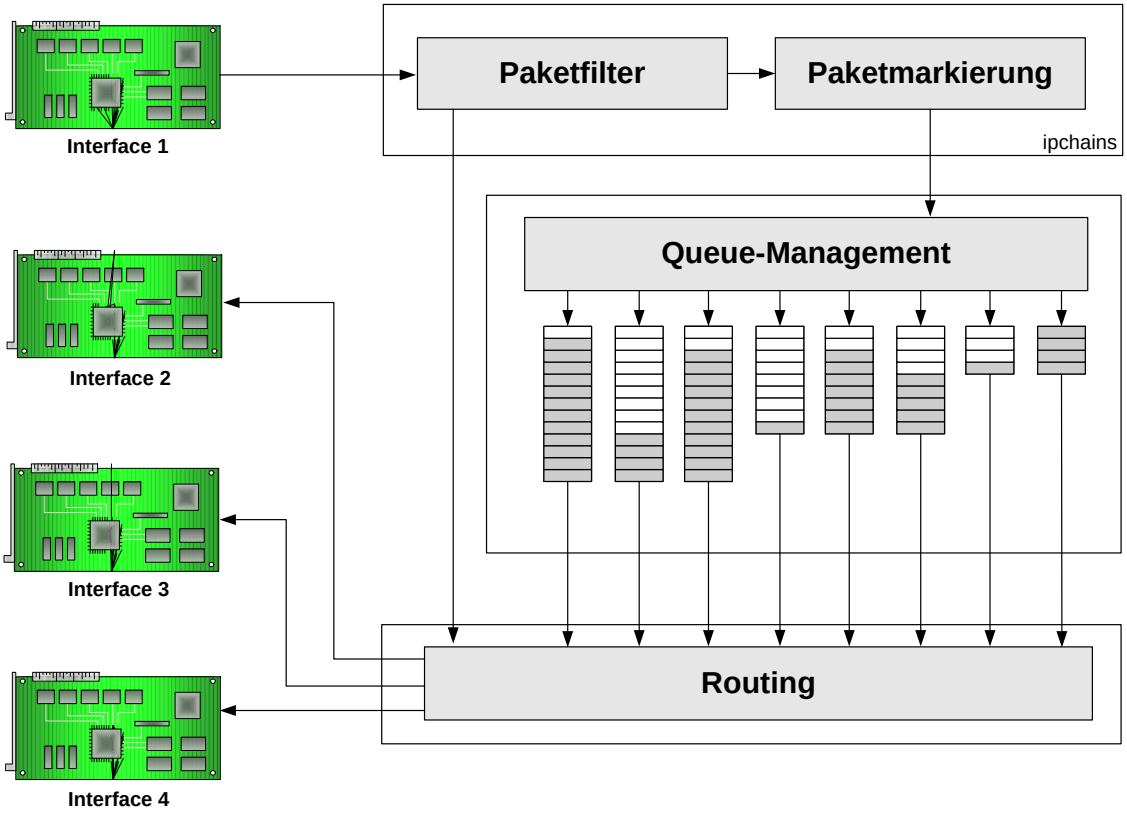


Abbildung 2.2: vereinfachte Darstellung der Blockstruktur des Linux Traffic Controls

**Interface 1** Dient als Incoming und Outgoing Interface zum RZ. Hier gehen die für das *QoS* relevanten Pakete durch.

**Interface 2** Netz-Block für die Flach-bauten im GHB (141.28.224.0/24)

**Interface 3** Netz-Block für den 2er Turm im GHB (141.28.227.0/24)

**Interface 4** Netz-Block für den 9er Turm im GHB (141.28.228.0/24)

### 2.2.1 Konzept 1

Um ein *QoS* zu implementieren wäre es von nöten jeglichen Traffic der das Shaping im RZ betrifft (Alles bis auf Traffic zum ASK-Wohnheim) mit in Betracht zu ziehen. Daher wäre der logische Ansatz auf dem Interface 1 in der Abbildung 2.2 die gesamten Klassen

zu implementieren. Dieses externe Interface bietet die Möglichkeit genau jenen Traffic zu begrenzen.
Dabei wäre folgendes Konzept sinnvoll:

1. Root-Class auf 1MBit begrenzt
2. Vererbung auf Unterklassen die sich nach der Tabelle 1.3 weiter unterteilen in Priorität und Bandbreite.
3. Vernachlässigen von ASK Traffic

Zu diesem Zwecke empfiehlt sich als QoS-Filter das "firewall-mark" Kernel Modul. Mittels diesem Modul kann man im Packet-Filter (in diesem Fall ipchains) die Pakete mit einem $2^(32)$ Bit Flag markieren. Bei der Definition der Klassen für die *traffic control* Filter werden dann die in der Firewall markierten Pakete den Entsprechenden Queueing Strategien zugewiesen.
Dadurch das ipchains verwendet wird kann auf die bestehende Struktur eingegangen werden. Änderungen im QoS müssen dann nur durch ein reload einer *chain* im Packet-Filter ausgeführt werden. Dies ist z.b. der Fall wenn ein Benutzer im GHB sein Traffic-Limit überschreitet. Dann wird er automatisch einer Shaping Klasse zugewiesen die ihm 1kb/s (10kbit) gewährt.

### 2.2.2 Konzept 2

Wie noch festgestellt wird hat Linux die Einschränkung nur ausgehenden Traffic limitieren zu können. Deswegen kam es dazu mehrere Konzepte zu erarbeiten. Im zweiten Konzept wurde versucht den Traffic auf den einzelnen Devices nur ausgehend zu betrachten. So ist Traffic der aus den Interfaces 2-3 rausgeht mit einer Source Adresse die nicht zum GHB Netz-Bereich gehört zu shapen. Hierzu muß aber auf jedem Device eine Queueing Strategie implementiert werden. Diese sind unabhängig voneinander weswegen alle 3 Interfaces zusammen 1MBit/s ergeben müssen. Hierbei wird die Gesamt-Bandbreite in der Annahme das alle 3 Interfaces die gleiche Gewichtung haben zu je $\frac{1}{3}$ bei der Initialisierung geteilt. In Spitzen-Zeiten wäre die Auslastung aller Interfaces dann gerecht geteilt.

### 2.2.3 Konzept 3

Das Konzept 2 hat den Nachteil das bei geringer Auslastung ein Benutzer nicht die maximal mögliche Bandbreite erreichen kann. Sein Maximum würde dann immer bei $\frac{1}{3}$ der möglichen Bandbreite liegen. Outgoing dagegen wären wiederum die gesamte Bandbreite möglich da das Outgoing Interface den gebündelten Traffic aus allen 3 internen

Interfaces erhält.
Die Lösung hierfür liegt in der rekursiven Möglichkeit bei Linux. Als erste Strategie wird dann ein Route-Filter die Pakete in Unterklassen auf den einzelnen Devices aufteilen. Dadurch wäre die Aufteilung der maximalen Bandbreite dynamisch durch den Route-Filter definiert. Die Route-Klassen können gegenseitig voneinander Traffic „borgen“ sofern sie selber nicht ihre Maximum benötigen. Die weiteren Klassen auf den einzelnen Interfaces haben als root-Klasse die einzelnen route-Klassen. Dadurch vererben sich auch jede weiteren Parameter. Vorteil für den Admin wäre das er auf jedem Device seine Klassen so definieren kann als ob es nur ein Device gibt. Das Load-Balancing und die Gewichtung übernimmt die route-Klasse.

## 2.3 Technische Einschränkungen

Der Linux-Kernel in der Version 2.4.4 bietet grundsätzliche Funktionen im Bereich von QoS, allerdings mit eingeschränkten Möglichkeiten.
Wie schon im Kapitel 2.2 beschrieben kann Linux nur ausgehenden Traffic beschränken. Auf eingehenden Traffic kann es keine Queue's aufbauen die ebenfalls verzögern. Hauptsächlich hängt das davon ab das wenn die Pakete die Filter erreichen schon im Kernel aus den Incoming-Queue's ausgetreten sind. In meinen Augen eine schlichtweg schlecht durchdachte Kernel Implementation. Denn wenn zum zuweisen der Pakete das Paket die Incoming Queue's schon verlassen hat, kann es auch einfach nicht mehr in eine Incoming Queue gelangen um dort eine Verzögerung zu bekommen.

Problematisch ist die Implementierung der „classes“ für QoS innerhalb des Kernels. Klassen und damit QoS können nur abhängig vom Interface konfiguriert werden. Daher muß man über den Umweg der “Routing Realms“ und “*tc* filter route“ gehen. Dies ermöglicht ein einigermaßen sinnvolles dynamisches Aufteilen zwischen den Interfaces. Es ist wohl durch die Struktur im Kernel nicht ganz einfach die Klassen so definieren zu können das sie auch andere Interfaces mit einbeziehen. Die nötigen Strukturen dazu sind im Kernel schon an ihr Device gebunden. Und Querverweise auf Speicherbereiche anderer Interfaces würde wohl zu einem Synchronisationsproblem führen.

Das Route-Konzept ist ebenfalls nicht ausgereift. So wurden Pakete innerhalb des Netzes 141.28.0.0/16 trotz route-tables nicht richtig in der Queueing Strategie zugeordnet. FH-Fremde Netz-Bereiche dagegen wurden richtig erkannt. Der Grund für dieses Fehlverhalten im Kernel konnte nicht gefunden werden.

Ebenfalls setzt das QoS nicht auf Protokolle auf die kein Connection-Flow Konzept ha-

ben. UDP ist ein sehr aggressives Protokoll das immer mit dem höchst möglichen Durchsatz sendet und Packet-Verluste in Kauf nimmt. Somit erreichen diese UDP Pakete schon den RZ Router mit maximalem Durchsatz und werden dann erst im GHB am Router verworfen. Für den Benutzer ändert sich nichts dadurch. Er wird auch UDP Pakete verzögert empfangen bzw bei einer vollen Klasse werden seine UDP Pakete gedropt.
Die Konzeption muß dieser Einschränkung Rechnung tragen.

## 2.4 Einsatzfähige Lösung

Als einzig Einsatzfähige Lösung bleibt mit diesen Einschränkungen nur noch das in Kapitel 2.2.2 beschriebene. Es bleibt der Nachteil bestehen das man nicht die maximal mögliche Bandbreite erhält. Um das Problem mit UDP einigermaßen einzuschränken wurden die NFS mounts zwischen *ftp.ai-lab.fh-furtwangen.de* und *ftp.ghb.fh-furtwangen.de* auf TCP umgestellt. Dies verhindert das die Shaping-Klasse im RZ Überfüllt wird ohne das auf dem GHB Router die Pakete ankommen.

# 3 Umsetzung

## 3.1 Konzept

Anforderung an die umzusetzende Lösung ist eine protokollabhängige Bandbreitenaufteilung und Priorisierung des zur routenden Traffics mit der Einführung von Klassen auf Basis der Spezifikation in Tabelle 1.3.
Abbildung 3.2 stellt das Klassenkonzept genauer dar: Auf dem Router werden mehrere Traffic-Klassen unterschiedlicher Priorität implementiert. In der Grafik stehen die wichtigen Protokolle an oberer Stelle, die zu vernachlässigbaren an unterer Stelle.

| **Class 5: ssh** | |
|---|---|
| Priorität: | 1 |
| max. Bandbreite: | 100 kbit/s |
| Typ: | isolated |

Abbildung 3.1: QoS-Klasse

Jede Klasse ist wie in Abbildung 3.1 dagestellt spezifiziert mit Protokollart, Priorität, Bandbreite und Typ. Gemäß des Protokolltyps wird ein Paket einer Klasse zugeordnet. Die Routingentscheidung des Routers wird dann anhand der **Priorität** einer Klasse priorisiert.
Desweiteren wird eine Klasse mit den Parametern **Bandbreite** und **Typ** näher beschrieben. Pakete einer Klasse vom Typ „bounded“ sind in ihrer Bandbreite eingeschränkt und können sich - falls mehr Bandbreite benötigt wird als die Klasse konfiguriert ist - keine Bandbreite von Klassen leihen deren Bandbreiten Kontigent nicht voll ausgeschöpft ist. Klassen vom Typ „isolated“ verleihen unter keinen Umständen ungenutzte Bandbreite an andere Klassen. Eine Klasse ohne jegliche Beschreibung kann sich dagegen aus dem Bandbreitenkontigent anderer bedienen und bietet auch ungenutzten Kontingent an andere Klassen an.

## 3.2 Kernel

Für den Einsatz von QoS musste der Linux-Kernel des Routes „wurmloch“ mit zusätzlichen Optionen neu compiliert werden. Im Anhang A.1 finden sich die relevanten Zeilen aus dem Konfigurationsfile, die zum Einbinden der nötigen Komponenten notwendig s. Auf dem GHB-Router kommt mit der umgesetzten Lösung ein Kernel in der Version 2.4.6 zum Einsatz.

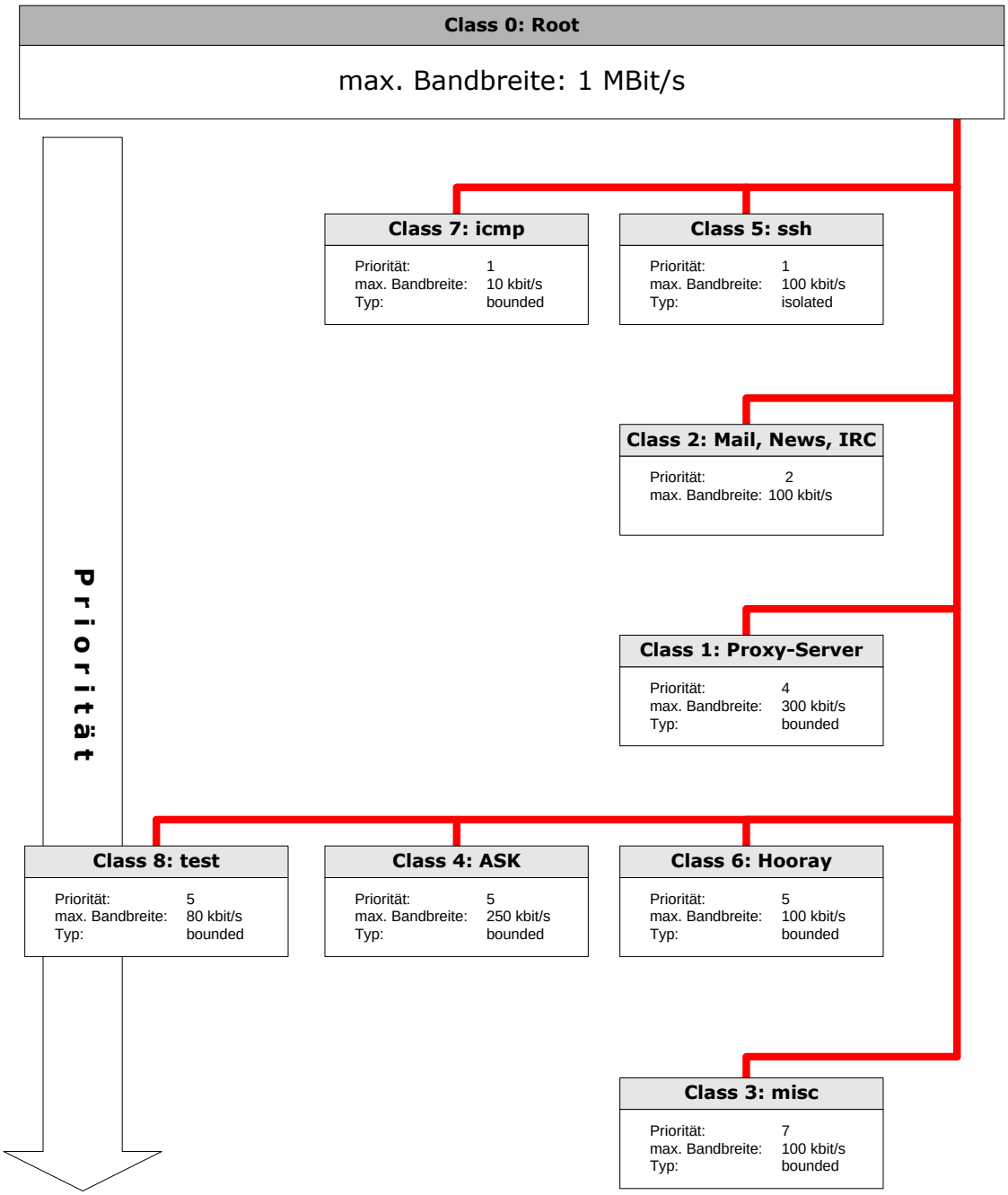


Abbildung 3.2: Klassenkonzept der Trafficprioisierung und -aufteilung

## 3.3 Init-Scripte

QoS wird über im Anhang A.2 zu findende Init-Script nach dem Booten des Routers initialisiert und gestartet, aber auch genauso gestoppt werden kann.

Dieses Script richtet das nach Abbildung 3.2 dargestellte Klassenkonzept innerhalb des Kernels (Zeilen 81 - 111) sowie weitere Klassen für Sonderzwecke (Zeilen 113-135) auf jedem Interface des Routers ein.

## 3.4 ipchains

Wie in Abbildung 2.2 dargestellt übernimmt ipchains übernimmt die Markierung der Pakete abhängig vom Protokoll. Das in Kapitel A.3 aufgeführte Script übernimmt die Einrichtung der entsprechenden Regelsätze nach der Spezifikation in 1.3. Das Script bindet die Dateien „/usr/local/etc/header/qosausnahmen“ und „/usr/local/etc/ghbshapeQoSload“ mit ein.
/usr/local/etc/header/qosausnahmen definiert eine Liste von Rechnern, deren in keiner Weise von QoS behandelt werden soll.
/usr/local/etc/ghbshapeQoSload richtet ein Shaping auf 1kBit/s für die User ein, die innerhalb von 7 Tagen zuviel Traffic erzeugt haben und daher zwangsweise gedrosselt werden. Dieses Script wird alle 10 min über ein automatisiertes Script neu erstellt.

# 4 Fazit

Das Projektziel der Einführung von QoS wurde erreicht.

Die Einführung von QoS im Netz des Studentenwohnheimes GHB zeigte deutliche Verbesserungen in bezug auf Anwendungen wie ssh. Die in Abbildung 4.1 dargestellte Messung der Paketlaufzeiten mit aktivem QoS und unter den gleichen Rahmenbedingungen wie bei der in Abbildung 1.2 dargestellten Messung zeigt die deutliche Verbesserung. Eine konstante Round-Trip-Time für ssh-Traffic von ca. 100 ms ist nun über den ganzen

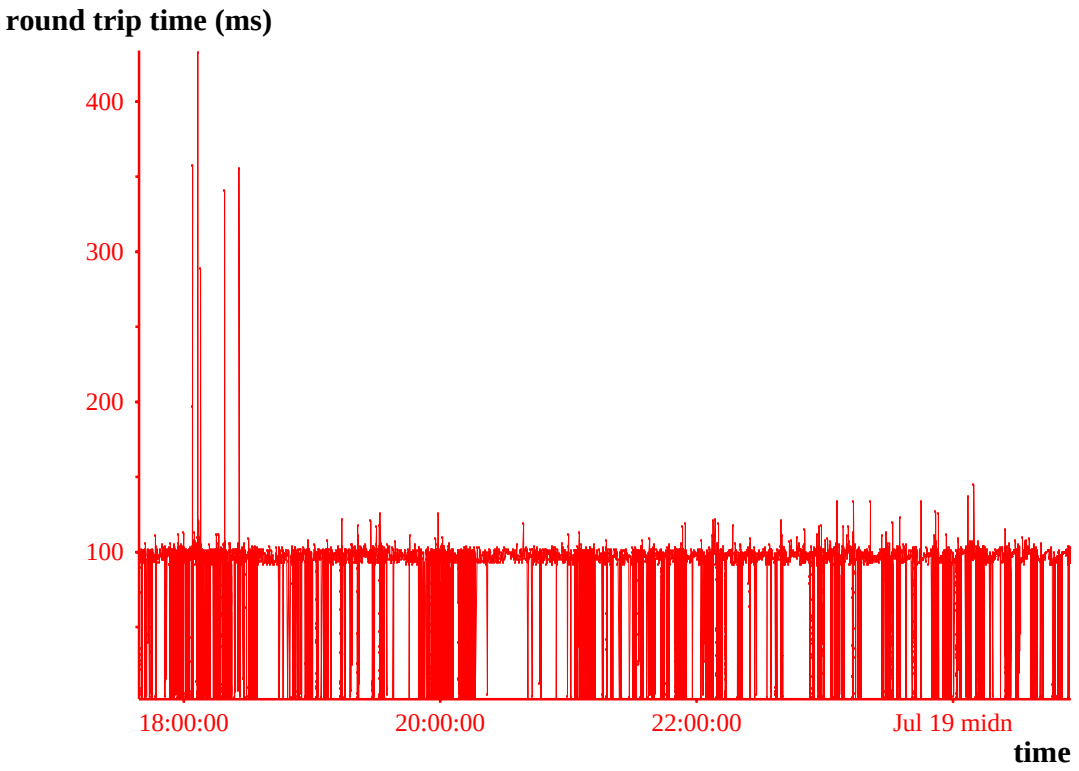


Abbildung 4.1: Paketlaufzeiten (Round-Trip-Time) einer ssh-Verbindung von realworld.ghb.fh-furtwangen.de nach father.foo.fh-furtwangen.de am 18. Juni 2001 zwischen 18:00 und 0:00 Uhr mit Einsatz von QoS

Tag hinweg möglich.
Ausserdem konnte „politisch kritischer“ Traffic wie Downloads von Audio- und Videodateien via FTP über die Parameter Bandbreite und Priorität stärker beeinflusst werden.

Das Shaping von Anwendern, die übermässig viel Traffic innerhalb einer definierten Zeitspanne erzeugt haben, konnte einfacher gestaltet werden. Die bisherige Lösung über eigene Shapingdevices konnte durch weitere QoS-Klassen mit niedrigen Parametern ersetzt werden.

Ausgereift ist die Lösung jedoch noch nicht. Die Beschränkung auf ledigliche Behandlung von ausgehenden Traffic auf einem Interface und das Erzeugen von interfaceabhängigen Klassen führt in Sonderfällen zu einem Versagen von QoS.
Ein Update der installierten Lösung bei weiterentwickelter Software zu einem späteren Zeitpunkt sollte daher unbedingt durchgeführt werden.

# A Konfigurationsfiles

## A.1 Kernelconfig Linux-2.4.6

Anmerkung: nur die relevanten Zeilen sind aufgeführt

Kernelconfig

```
#
# QoS and/or fair queueing
#
CONFIG_NET_SCHED=y
CONFIG_NETLINK=y
CONFIG_RTNETLINK=y
CONFIG_NET_SCH_CBQ=y
CONFIG_NET_SCH_CSZ=m
CONFIG_NET_SCH_PRIO=m
CONFIG_NET_SCH_RED=y
CONFIG_NET_SCH_SFQ=y
CONFIG_NET_SCH_TEQL=m
CONFIG_NET_SCH_TBF=y
CONFIG_NET_SCH_GRED=m
CONFIG_NET_SCH_DSMARK=m
CONFIG_NET_SCH_INGRESS=m
CONFIG_NET_QOS=y
CONFIG_NET_ESTIMATOR=y
CONFIG_NET_CLS=y
CONFIG_NET_CLS_TCINDEX=y
CONFIG_NET_CLS_ROUTE4=y
CONFIG_NET_CLS_ROUTE=y
CONFIG_NET_CLS_FW=y
CONFIG_NET_CLS_U32=y
CONFIG_NET_CLS_RSVP=m
CONFIG_NET_CLS_RSVP6=m
CONFIG_NET_CLS_POLICE=y
```

## A.2 /usr/local/sbin/qos

Das Init-Script erzeugt innerhalb des Kernels die QoS-Klassen mit den spezifizierten Werten.

/usr/local/sbin/qos

```
#!/bin/sh

add_class() {
#  $1=interface $2= parent class $3 classid $4 hiband
# $5 lowband $6 handle $7 prio $8 $9 style


        if=‘echo $1 | cut -c4‘

        echo "addclass on $1: addclass $2 $3 $4 $5 \
        ‘expr $if + 1‘$6 $7 $8 $style2"

        tc class add dev $1 parent $2 classid $3 \
                cbq bandwidth 1mbit rate $4 allot 1514 \
                weight $5 prio $7 maxburst 10 avpkt 1000 \
                $8 $style 2

        tc qdisc add dev $1 parent $3 sfq quantum 1514b \
          perturb 15
        tc filter add dev $1 protocol ip prio $7 \
                handle ‘expr $if + 1‘$6 fw class id $3
}


case "$1" in

start)
        ;;

status)

        echo;echo; echo "========= Queing Disc ========";
        echo;echo
        tc -s -d qdisc show dev $2
```

```
        echo;echo; echo "========= Class ========";
        echo;echo
        tc -s -d class show dev $2
        exit 0
        ;;

stop)
        echo "flushing qos-classes on device eth0 ..."
        tc qdisc del dev eth0 root
        echo "flushing qos-classes on device eth1 ..."
        tc qdisc del dev eth1 root
        echo "flushing qos-classes on device eth2 ..."
        tc qdisc del dev eth2 root
        echo "flushing qos-classes on device eth3 ..."
        tc qdisc del dev eth3 root
        echo "flushing qos-classes on device eth4 ..."
        tc qdisc del dev eth4 root
        exit 0
        ;;

        *)
        echo "Usage: $0 {start|stop|status <device>}"
        exit 1
        ;;
esac

# here we go ...


for i in ‘ifconfig -a |grep "^[eth]"| cut -d" " -f1,0‘
do

        #####
        # flushing the rules
        tc qdisc del dev $i root 2>&1 | grep -v "No such file"


        #####
        # setting root-classes
        tc qdisc add dev $i root handle 10: cbq
```

```
            bandwidth 1mbit avpkt 1000
         tc class add dev $i parent 10:0 classid 10:1 cbq \
            bandwidth 1Mbit rate 100kbit allot 1514 \
            weight 10kbit prio 8 maxburst 10 avpkt 100 bounded

         #####
         # setting sub-classes
         #
         # call:
         # addclass interface parent_class classid hiband \
         #          lowband handle prio styles

         #class 1: proxy
         add_class $i 10:1 10:100 600kbit 60Kbit 01 4 bounded

         #class 2: mail, news, irc
         add_class $i 10:1 10:200 100kbit 10kbit 02 2

         #class 3: misc
         add_class $i 10:1 10:300 200kbit 20kbit 03 7 bounded

         #class 4: ask
         add_class $i 10:1 10:400 250kbit 25kbit 04 5 bounded

         #class 5: ssh
         add_class $i 10:1 10:500 100kbit 10kbit 05 1 isolated

         #class 6: hooray-wg's
         add_class $i 10:1 10:600 100kbit 10kbit 06 5 bounded

         #class 7: icmp
         style2=isolated
         add_class $i 10:1 10:700 10kbit 1kbit 07 1 bounded
         style2=""
         #class 8: test shape
         add_class $i 10:1 10:800 80kbit 8kbit 08 5 bounded

#### shaping für ghb user die uebermaessig viel traffic
     produzieren (1kbyte/s)

```

```
        add_class $i 10:1 10:1001 10kbit 1kbit 70 8 bounded
        add_class $i 10:1 10:1002 10kbit 1kbit 71 8 bounded
        add_class $i 10:1 10:1003 10kbit 1kbit 72 8 bounded
        add_class $i 10:1 10:1004 10kbit 1kbit 73 8 bounded
        add_class $i 10:1 10:1005 10kbit 1kbit 74 8 bounded
        add_class $i 10:1 10:1006 10kbit 1kbit 75 8 bounded
        add_class $i 10:1 10:1007 10kbit 1kbit 76 8 bounded
        add_class $i 10:1 10:1008 10kbit 1kbit 77 8 bounded
        add_class $i 10:1 10:1009 10kbit 1kbit 78 8 bounded
        add_class $i 10:1 10:1010 10kbit 1kbit 79 8 bounded
        add_class $i 10:1 10:1011 10kbit 1kbit 80 8 bounded
        add_class $i 10:1 10:1012 10kbit 1kbit 81 8 bounded
        add_class $i 10:1 10:1013 10kbit 1kbit 82 8 bounded
        add_class $i 10:1 10:1014 10kbit 1kbit 83 8 bounded
        add_class $i 10:1 10:1015 10kbit 1kbit 84 8 bounded
        add_class $i 10:1 10:1016 10kbit 1kbit 85 8 bounded
        add_class $i 10:1 10:1017 10kbit 1kbit 86 8 bounded
        add_class $i 10:1 10:1018 10kbit 1kbit 87 8 bounded
        add_class $i 10:1 10:1019 10kbit 1kbit 88 8 bounded
        add_class $i 10:1 10:1020 10kbit 1kbit 89 8 bounded

done
exit 0
```