Technische Universität Chemnitz

– Lehrmaterial –

Studienarbeit QoS and/or fair Queuing

© Jan Horbach

25. November 2001

Inhaltsverzeichnis

1. Aufgabenstellung **5**

2. QoS und Linux **7**

3. Hinweise zum Praktikum **11**
3.1. Versuchsaufbau 11
3.2. Konfiguration der Systeme 13
3.3. Software zur Auswertung der Resultate 16
3.4. Hilfsskripte 18

4. Versuchsanleitung mit Musterlösungen **21**
4.1. Versuchsvorbereitung 21
4.1.1. Was ist QoS? 21
4.1.2. Grundlegende Verfahren 24
4.1.3. Realisierung in Linux 32
4.2. Versuchsumgebung 42
4.2.1. Versuchsaufbau 42
4.2.2. Zur Verfügung stehende Software 44
4.3. Versuchsdurchführung 48
4.3.1. Versuch: Verschiedene Klassen mit CBQ 48
4.3.2. Bemerkungen zum Versuch 1 50
4.3.3. Versuch: Borgen von Bandbreite mit CBQ 52
4.3.4. Bemerkungen zum Versuch 2 54
4.3.5. Versuch: Vermeidung von Slow Starts mit RED 57
4.3.6. Bemerkungen zum Versuch 3 59
4.3.7. Versuch: Kanalbündelung mit TEQL 61
4.3.8. Bemerkungen zum Versuch 4 62

5. Abschließende Bemerkungen **63**

6. Quellen **65**

A. Listings **67**
A.1. Von Seite 18 (3.4): Vollständiger Quellcode zu tcc 67
A.2. Von Seite 18 (3.4): Vollständiger Quellcode zu netpipe 71

A.3. Von Seite 19 (3.4): Vollständiger Quellcode zu routes 72
A.4. Von Seite 19 (3.4): Vollständiger Quellcode zu prak 74

Lernziele:
Ziel dieses Versuches ist es, die Möglichkeiten des Linux-Kerns hinsichtlich Quality of Service kennenzulernen und anzuwenden. Dabei sollen die verschiedenen Verfahren vorgestellt und praktisch eingesetzt werden.

1. Aufgabenstellung

Auf der Basis der Studienarbeit *"Gigabit Ethernet Praktikum"* sind die neuen experimentellen Möglichkeiten des Linux-Kerns ab 2.1 zum *QoS and/or fair queuing* in das Praktikum einzubauen.

Die verschiedenen Varianten sollen zunächst untersucht und auf ihre praktische Anwendbarkeit getestet werden. Da es sich um experimentelle Eigenschaften des Kerns handelt, sollen Aussagen zur Stabilität getroffen werden.

Ziel der ersten Etappe ist es, einen Vorschlag zur Einarbeitung dieser neuen Steuerungsmechanismen in das Praktikum *"Gigabit Ethernet"* zu erarbeiten.

In der 2. Etappe soll die Erweiterung des Praktikums zu diesen Themen erfolgen. Dabei ist insbesondere auf die Umsetzung der Kenntnisse aus den Vorlesungen *Modellierung und Simulation* zu achten.

Betreuer

- Dr. Ing. Jörg Anders
- Prof. Dr.-Ing. habil. Uwe Hübner

2. QoS und Linux

Quality of Service dient dazu, verschiedene Serviceklassen unterschiedlich zu behandeln. Die Klassen werden dabei nach Protokoll, Port, Quell- bzw. Zieladresse u.a. unterschieden. Die Behandlung erfolgt im Moment nur für ausgehenden Traffic, sie umfasst z.B. die unterschiedliche Vergabe von Bandbreiten und Prioritäten, aber auch Algorithmen zur Überlastvermeidung von TCP-Datenströmen und zur Verteilung von Traffic auf mehrere Netzinterfaces. Eine detaillierte Übersicht der existierenden Verfahren befindet sich in der Versuchsanleitung.

Linux unterstützt eine Reihe dieser Verfahren, von einfachen Queuing-Algorithmen bis hin zu Differentiated Services. Als leistungsfähigster Algorithmus ist hier Class-Based Queuing (CBQ) zu nennen, der die Unterteilung des Traffics in verschiedene Klassen ermöglicht. Das Management erfolgt über das Kommandozeilen-Tool Traffic Control (TC). Auch dazu gibt es einen umfangreichen Abschnitt in der Versuchsanleitung.

Aussagen zur Stabilität

Die in den Versuchen verwendeten Algorithmen zeigten ein sehr gutes Verhalten hinsichtlich Stabilität unter Volllast, aber auch im Langzeitversuch (ich persönlich setze CBQ mit verschiedenen Bandbreiten für unterschiedliche Protokolle und Zielrechner seit ca. 4 Monaten ein). QoS lässt sich auch während einer laufenden Übertragung einfach ein- und auch wieder ausschalten, ohne dass der Datenstrom gestört wird. Da QoS eventuell auch bald im CSN benutzt werden soll, wurde ich gebeten zu testen, wie sich CBQ mit sehr vielen Klassen verhält. Dazu habe ich folgendes kleine Skript verwendet:

```
#!/bin/bash

tcc cbq    eth0 1: root 100MBit
tcc class  eth0 1:1 1:0 100MBit 100MBit 10MBit 8

for i in `seq 2 $1`; do
  tcc class  eth0 1:$i 1:1 100MBit ${i}0KBit ${i}KBit 5 bounded
  tcc tbf    eth0 $i:  1:$i        ${i}0KBit
  tcc filter eth0 1:$i 1:0 match ip dport $((${i}+1000)) 0xffff
done
```

Das Skript legt eine frei spezifizierbare Anzahl von Klassen an, beschränkt die Bandbreite für jede dieser Klassen (Nummer der Klasse ∗10 KBit/s) und ordnet den Traffic, der zu einem bestimmten Port geht (Nummer der Klasse + 1000) der entsprechenden Klasse zu. Experimentiert habe ich mit 2000 und 5000 Klassen, indem ich mit NetPIPE eine Messreihe an einigen ausgewählten Ports gestartet habe. Da die Bandbreite vom Port abhängig ist ((Port - 1000) ∗10 KBit/s), müsste eine unterschiedliche Bandbreite festzustellen sein.

Schon bei 2000 Klassen ist der Overhead, der beim Zuordnen zu den Klassen entsteht, deutlich zu spüren: Erst einmal dauert es sehr lange, diese Klassen anzulegen (ca. 25min bei 5000). Der Kernel beschwert sich ab etwa der 150. Klasse, dass die Klassen ein falsches Quantum von 0 haben, dass er das aber in Ordnung gebracht hat. Und es werden maximal 14-15MBit/s erreicht, obwohl ich Klassen getestet habe, die auf 10, 20, 30 und 50MBit/s festgelegt waren. Zusätzlich gab es Einbrüche auf 5-8MBit/s bei Blockgrößen, die Vielfache von 8KByte waren.

> *Eventuell hätte sich daraus ein weiterer Versuch aufbauen lassen können. Da dieser aber nur aus Messungen bestanden und darüber hinaus auch noch sehr lange gedauert hätte, habe ich davon abgesehen.*

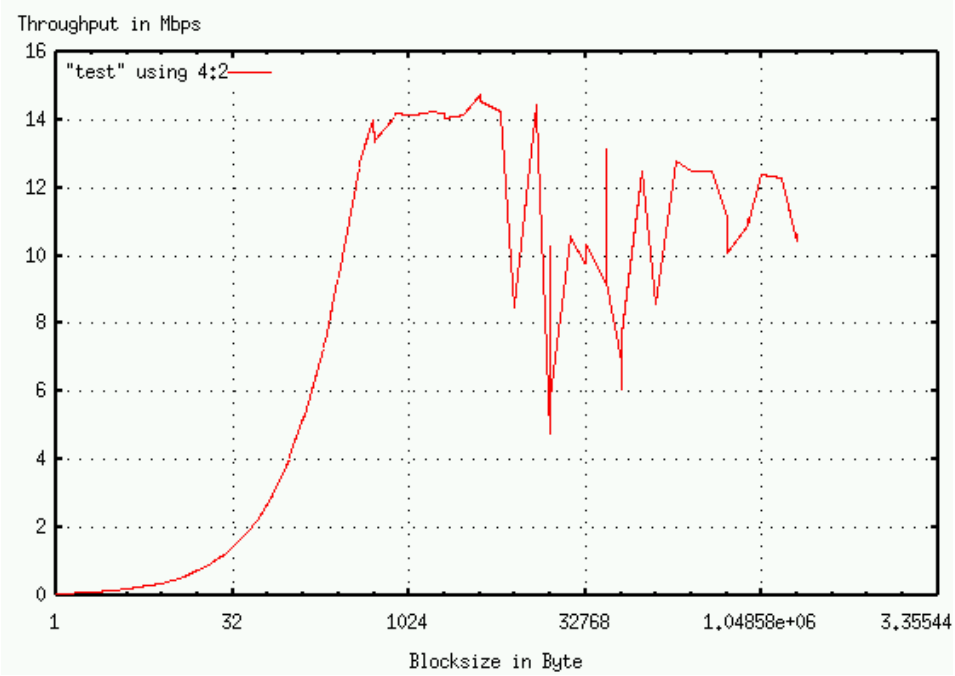

Abbildung 2.0-1.: CBQ mit 2000 Klassen

3. Hinweise zum Praktikum

3.1. Versuchsaufbau

Für die Entwicklung der Versuche stand mir ein Labor mit vier Rechnern zur Verfügung, die drei miteinander verbundene Netzwerke bildeten:

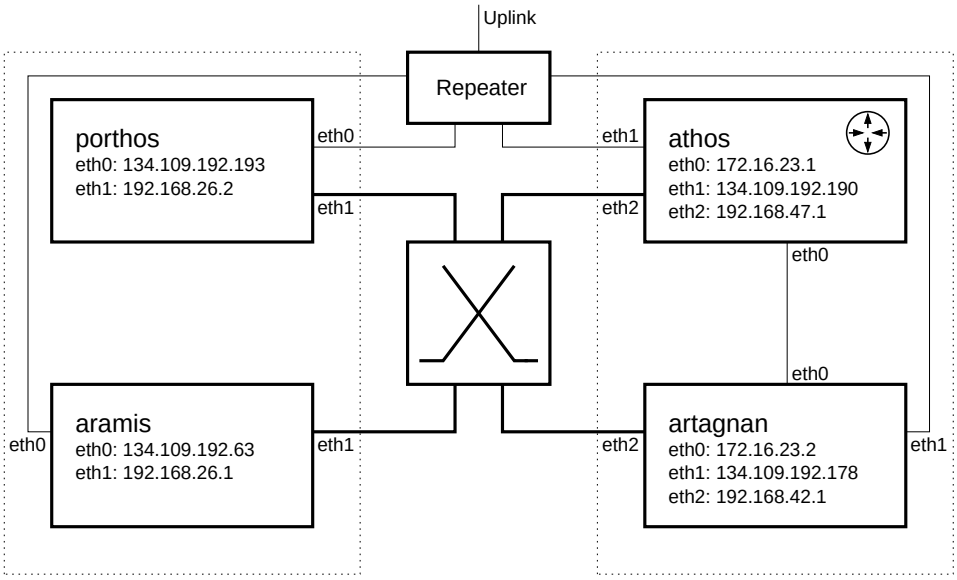

Abbildung 3.1-1.: Versuchsaufbau

Das Gigabit-Netz spielt für die QoS-Versuche keine Rolle, allerdings wurden die Versuche noch in diesem Netz entwickelt. Da aber in Zukunft mehrere Labors eingerichtet werden sollen, ist es unpraktikabel, in jedes einen teuren Gigabit-Switch zu stellen. Außerdem lassen sich in einem 100MBit-Netz wesentlich leichter Überlastsituationen herstellen, die wesentlich für die meisten Versuche sind. Deshalb existiert noch das 100MBit-Netz mit den offiziellen 134.109-er IP-Adressen. Die Rechner sind hier mit einem 10/100MBit-Repeater verbunden, der auch einen Uplink zum Uninetz besitzt. Dieser sollte aber bei der Versuchsdurchführung gekappt werden, um nicht das gesamte Subnetz zu beeinträchtigen. Zusätzlich existiert noch eine 100MBit-Direktverbindung zwischen *athos*

und *artagnan*, um eine Routingmöglichkeit zwischen verschiedenen Netzen zu haben und die gleichzeitige Nutzung mehrerer Netzkarten zu zeigen (Versuch *Kanalbündelung mit TEQL*).

Da die Gigabit-Möglichkeiten des Labors nicht mehr genutzt werden, sind die Versuche kein Bestandteil des *Gigabit Ethernet Praktikums*, sondern stellen eine eigene Versuchsreihe dar, die zur Zeit aber noch im selben Labor stattfindet. Wenn die Versuche irgendwann in einem anderen Labor laufen sollen, muss die Versuchsanleitung hinsichtlich Aufbau, IP-Adressen usw. überarbeitet werden. Falls mehrere Labors genutzt werden sollen, wäre es optimal, wenn man für das 100MBit-Netz zwischen allen Rechnern auch noch einen privaten IP-Adressraum verwendet, damit die Versuchsanleitung für alle Labors identisch sein kann. Ein Rechner sollte aber auch eine Verbindung nach außen besitzen (möglichst mit DNS), damit man bei Unklarheiten auch eine externe Dokumentation zu Rate ziehen kann.

3.2. Konfiguration der Systeme

Im Falle eines Neuaufbaus des Praktikums müssen folgende Einstellungen an den verwendeten Computern vorgenommen werden, damit alles unterstützt wird und vom System keine Störungen (z.B. durch ein Update der Locate-Datenbank) auftreten:

Dienste deaktivieren

- alle unnötigen Dienste in `/etc/inetd.conf` deaktivieren (notwendig ist nur FTP)
- alle unnötigen Dienste aus `/etc/rc.config` entfernen (SuSE Linux)
- evtl. nicht benötigte Links aus den Runlevels auf die Startup-Skripte löschen
- CRON-Aktivität auf ein Minimum reduzieren (`/etc/rc.config` und `/etc/cron.daily`)
- DNS deaktivieren (`/etc/resolv.conf`), da SSH sonst vergeblich einen Name-Lookup versucht

Kernelkonfiguration

Für das Praktikum habe ich den Kernel 2.2.16 eingesetzt, da er bereits alle benötigten Algorithmen enthält und der 2.4-er Kern noch nicht die Gigabit-Netzwerkkarten unterstützt. Da diese jedoch für den Versuch irrelevant sind, kann bei Bedarf auch ein neuerer Kernel eingesetzt werden, mit dem dann noch mehr Verfahren zur Verfügung gestellt werden. In den Netzwerkeinstellungen sollten die Rechner als *Advanced Router* konfiguriert werden. Die dazugehörigen Einstellungen sind ebenfalls auszuwählen, unbedingt notwendig sind die Firewalling-Punkte. *Optimize as router not host* kann man ebenfalls auswählen, das wurde aber nicht getan, da der Einfachkeit halber der selbe Kernel für alle vier Rechner verwendet wurde. Die Einstellungen zum *QoS and/or fair queueing* sind alle vorzunehmen, soweit möglich, als Modul.

IP-Forwarding im Router

Damit der Router *athos* auch Pakete weiterleiten kann, ist in `/etc/rc.config` die Variable IP_FORWARD auf "yes" zu setzen. Das kann aber auch (z.B. bei Verwendung einer anderen Linux-Distribution) durch Einfügen von `echo "1" > /proc/sys/net/ipv4/ip_forward` in eines der Startup-Skripte erreicht werden.

FTP-Nutzer in /etc/ftpusers

Hier muss der Eintrag *root* gelöscht werden, um auch dem Root-Nutzer FTP zu ermöglichen. Alternativ wäre auch denkbar, für FTP-Transfers extra Nutzer anzulegen und diese zu verwenden.

Hostnamen in /etc/hosts

Da kein Nameserver zur Verfügung steht, müssen alle Rechner, Switch-Adressen und IP-Adressen virtueller Interfaces ins Hosts-File eingetragen werden:

```
127.0.0.1       localhost
134.109.192.63  aramis.informatik.tu-chemnitz.de aramis
134.109.192.178 artagnan.informatik.tu-chemnitz.de artagnan
134.109.192.190 athos.informatik.tu-chemnitz.de athos
134.109.192.193 porthos.informatik.tu-chemnitz.de porthos
172.16.23.1     athos.megabit.org
172.16.23.2     artagnan.megabit.org
192.168.26.1    aramis.gigabit.org
192.168.26.2    porthos.gigabit.org
192.168.26.253  slb26.gigabit.org slb26
192.168.26.254  switch26.gigabit.org switch26
192.168.42.1    artagnan.gigabit.org
192.168.42.253  slb42.gigabit.org slb42
192.168.42.254  switch42.gigabit.org switch42
192.168.47.1    athos.gigabit.org
192.168.47.253  slb47.gigabit.org slb47
192.168.47.254  switch47.gigabit.org switch47
```

Die Adressen des Netzes *gigabit.org* können natürlich entfallen, wenn das Praktikum in einem anderen Labor neu aufgebaut wird. Sie sind nur notwendig für das *Gigabit Ethernet Praktikum*.

Lokale Profileinstellungen in /etc/profile.local und /root/.bashrc

In `/etc/profile.local` ist folgendes einzutragen, um das angegebene Verzeichnis dem Pfad hinzuzufügen:

```
PATH=$PATH:/root/praktikum
export PATH
```

Da dieses File nur von einer Login-Shell ausgewertet wird, ist in `/root/.bashrc` noch folgende Zeile zu ergänzen, um `/etc/profile` auszuführen, die ihrerseits dann wieder `/etc/profile.local` lädt:

```
. /etc/profile
```

3.3. Software zur Auswertung der Resultate

Benötigte Software

Um die vorgenommenen Änderungen zu messen bzw. anderweitig zu überprüfen, muss folgende Software auf den Rechnern vorhanden sein:

- TC: zur Manipulation der QoS-Einstellungen
- FTP und SCP: für Filetransfers, Dateien zum Transfer sollten unter `/root/prak_files/` liegen
- NetPIPE und GnuPlot: für Messreihen und deren grafische Auswertung, der Pfad zu NetPIPE ist im Skript `netpipe` einzutragen
- TCP-Dump und Ethereal: zum Lauschen an den Interfaces, Ethereal ist nicht unbedingt notwendig
- IfConfig und Route: zur Anzeige und Manipulation von Interfaces und Routen
- Watch: um in bestimmten Intervallen z.B. IfConfig aufrufen zu können
- XV: um Screenshots machen zu können

Bis auf NetPIPE sind in der verwendeten Linux-Distribution (SuSE 6.4) alle Programme enthalten. Die verwendete Version von NetPIPE (2.3) kann unter folgender URL bezogen werden:

ftp://ftp.scl.ameslab.gov/pub/netpipe/netpipe-2.3.tar.gz

Das Archiv ist mit `tar xvzf netpipe-2.3.tar.gz` zu entpacken und im Verzeichnis `netpipe-2.3` ein `make TCP` und ein `mv NPtcp /usr/local/bin/` auszuführen, um das Programm zu compilieren und zu installieren.

Hinweise zur Handhabung der Software finden sich in der Versuchsanleitung.

Messreihen mit NetPIPE

NetPIPE (Network Protocol Independent Performance Evaluator) sendet und empfängt im Ping-Pong-Verfahren Datenblöcke, deren Größe stetig erhöht wird. Durch das Ping-Pong wird sichergestellt, dass sich immer nur ein Block auf der Leitung befindet. Somit werden Kollisionen nahezu ausgeschlossen (außer z.B. mit ARP). Ein Paper zu dem Programm und weitere Informationen finden sich unter folgender URL:

http://www.scl.ameslab.gov/netpipe/

Grafische Auswertung mit GnuPlot

Um die mit NetPIPE erzeugten Messfiles grafisch auszuwerten, kann GnuPlot verwendet werden. Bevor die Graphen gezeichnet werden, müssen mehrere Einstellungen vorgenommen werden. Das geschieht durch `load "/root/praktikum/ds.gnu"`. Das File `ds.gnu` steht für "Durchsatzgraph" und besitzt folgenden Inhalt:

```
set grid
set logscale x 2
set key left top Left noreverse box linetype -2 linewidth 1.000
samplen 4 spacing 1 width 0
set xlabel "Blocksize in Byte"
set ylabel "Throughput in Mbps"
set nologscale y
```

Die Messfiles werden dann mit `plot "MESSFILE1" using 4:2 w l,"MESSFILE2" using 4:2 w l, ...` gezeichnet, indem der Durchsatz (2. Spalte im File) über der Blockgröße (4. Spalte) abgetragen wird.

3.4. Hilfsskripte

Für das Praktikum existieren unter `/root/praktikum/` eine Reihe von Hilfsskripten, die einerseits den Studenten den Umgang mit den QoS-Möglichkeiten des Linux-Kerns erleichtern sollen, andererseits aber auch der Administration dienen.

/root/praktikum/tcc

Da die Handhabung des TC-Kommandos sehr komplex ist, gibt es für diesen Versuch eine vereinfachte Variante namens TCC (Traffic Control Changer), der auch gleich die benötigten Module mit lädt und folgende Syntax besitzt:

```
tcc cbq    DEVICE HANDLE { root | PARENTID } BANDWIDTH
tcc red    DEVICE HANDLE { root | PARENTID } BANDWIDTH PROBABILITY
tcc sfq    DEVICE HANDLE { root | PARENTID }
tcc tbf    DEVICE HANDLE { root | PARENTID } RATE
tcc teql   DEVICE
tcc class  DEVICE CLASSID PARENTID BANDWIDTH RATE WEIGHT PRIO \
           [ bounded ] [ isolated ]
tcc filter DEVICE FLOWID PARENTID match SELECTOR
tcc remove DEVICE
tcc list   DEVICE [ -s ]
```

Vollständiger Quellcode zu tcc (Siehe: A.1)

/root/praktikum/netpipe

Dieses Skript dient dazu, NetPIPE mit bestimmten voreingestellten Parametern zu starten, und zwar einer Puffergröße von 65536 Byte und einer Unter- bzw. Obergrenze von 1 bzw. 2097152 Byte für die Größe der übertragenen Blöcke. Der Sender und der Receiver werden mit folgenden Aufrufen gestartet:

```
netpipe send HOST PORT MESSFILE
netpipe recv PORT
```

Vollständiger Quellcode zu netpipe (Siehe: A.2)

/root/praktikum/routes

Das ist eher ein Hilfsskript, welches bei der Entwicklung der Versuche entstanden ist. Da die Routen für das *Gigabit Ethernet Praktikum* gesetzt sind, werden sie mit `routes set` für die QoS-Versuche neu und mit `routes reset` zurückgesetzt. Ein Aufruf von `routes list` listet die Routen auf allen vier Rechnern auf.

```
routes { set | reset | list }
```

Vollständiger Quellcode zu routes (Siehe: A.3)

/root/praktikum/prak

Dieses Skript dient nur der Administration der beteiligten Rechner. Mit `prak sync` werden bestimmte Dateien und Verzeichnisse (siehe Anhang) von *athos* auf die anderen Rechner übertragen, `prak pack` legt ein Archiv des Praktikumsverzeichnisses an, und `prak reboot` bzw. `prak halt` dienen dazu, alle Rechner auf einmal neu starten oder anhalten zu können.

```
prak { sync | pack | reboot | halt }
```

Vollständiger Quellcode zu prak (Siehe: A.4)

4. Versuchsanleitung mit Musterlösungen

4.1. Versuchsvorbereitung

4.1.1. Was ist QoS?

Der Begriff **Quality of Service (QoS)** besitzt viele verschiedene Bedeutungen und wird von verschiedenen Seiten unterschiedlich benutzt, was zu einiger Verwirrung geführt hat. Deshalb wurde 1997 auf dem Next Generation Internet (NGI) Workshop in Virginia eine (sehr allgemeine) Definition geprägt: QoS differenziert Traffic und Services, d.h. verschiedene Classes of Service (CoS) werden unterschiedlich behandelt.

Man stelle sich z.B. folgendes Szenario vor: In einem Campusnetz teilen sich eine Vielzahl von Studenten eine relativ begrenzte Netzanbindung. Zu bestimmten Zeiten wollen so viele Studenten das Netz nutzen, dass ein vernünftiges Arbeiten nicht mehr möglich ist. Ein Ansatz wäre die Anschaffung neuer Netztechnik, um die verfügbare Bandbreite zu erhöhen. Das könnte sogar soweit gehen, das Netz zu "over-engineeren", d.h. eine Bandbreite zur Verfügung zu stellen, die mindestens so groß wie die Summe der Bandbreiten pro Nutzer ist. Aus Kostengründen wird das aber so gut wie nie gemacht, sondern reale Netze sind in aller Regel "over-subscribed", wobei die Bandbreite nur für eine durchschnittliche Nutzung ausreichend ist. Falls das Netz nun aber stärker belastet wird als vorgesehen, befinden wir uns wieder bei der Ausgangssituation. Der Ausweg hieraus besteht aus einer differenzierten Behandlung von verschiedenen Traffic-Klassen. In unserem Campusnetz könnte z.B. Traffic, der von Forschung &Lehre herrührt, der Vorzug gegenüber dem diverser Netzwerkspiele gegeben werden.

Durch QoS kann z.B. auch garantiert werden, dass wichtige Management-Daten auch in einer Überlastsituation noch ankommen (indem sie bevorzugt behandelt werden), oder dass interaktive Verbindungen eine mehr oder weniger garantierte Antwortzeit bekommen, während das für simple Filetransfers nicht zwingend notwendig ist.

Was versteht man eigentlich unter den Begriffen Quality und Service?

Quality:

- verlässliche Datenlieferung, kein Datenverlust
- minimale Verzögerung
- konstante Verzögerungscharakteristika (z.B. konst. Jitter = Varianz zwischen min. und max. Verzögerung)
- effizienteste Nutzung von Netzwerkressourcen (kürzeste Entfernung usw.)

Service:

- End-zu-End-Kommunikation oder Client-Server-Applikationen
- von E-Mail bis Desktop Video, von Surfen bis Chatten
- Protokollumgebungen (IP, IPX, AppleTalk u.a.)
- hierarchisch (z.B. SAP-Typen innerhalb von IPX)

Classes of Service (CoS):

Um die verschiedenen Services unterschiedlich behandeln zu können, teilt man sie in Serviceklassen auf. Dabei werden sie z.B. nach folgenden Gesichtspunkten klassifiziert:

- Protokoll (IP, TCP, UDP, IPX, ...)
- Quell-/Zieladresse und/oder Quell-/Zielport (Flow = Adressen + Ports)
- Netzinterface der Quelle
- IP Precedence Bits (3 Bit) im TOS-Byte des IP-Headers
- IPv6: 4 Bit Priority/Class-Field + 24 Bit Flow-Label

Frage 4.1.1.1:
Was ist bei der Benutzung der IP Precedence Bits zur Klassifizierung von Datenströmen zu beachten, wenn die so klassifizierten Pakete ihren Weg durch das Netz nehmen? Denken Sie dabei daran, dass die Pakete unter Umständen viele verschiedene Netzabschnitte durchlaufen, die durch Router voneinander getrennt sind, die die Precedence Bits verschieden interpretieren oder meist gar nicht beachten.

Die Netzbetreiber müssen sich über die Bedeutung der Precedence Bits einigen, oder in den Routern muss eine "Übersetzung" stattfinden.

Probleme bei der Umsetzung von QoS

- Netzwerkcharakteristiken müssen vorhersagbar bleiben
- z.B. RTT (Round-Trip Time) → wichtig für TCP-Timeouts

- Filtering bevorzugt an Endpunkten → belastet Netzwerkkern nicht so sehr
- aber: dadurch auch leicht manipulierbar (keine Kontrolle während des Transfers)
- sehr eng mit Routing verbunden → evtl. dort einbauen
- aber: dynamisches Routing in Abhängigkeit von Last noch sehr unausgereift
- Problem wegen Asymmetrie (unterschiedliche Wege für Hin- und Rückweg)
- QoS-Maßnahmen machen sich meist nur in Überlastsituationen bemerkbar
- Messungen verfälschen Ergebnis zusätzlich

Vertiefung:

Paul Ferguson, Geoff Huston:

"Quality of Service - Delivering QoS on the Internet and in Corporate Networks"
http://www.wiley.com/compbooks/catalog/24358-2.htm

Wiley Computer Publishing, 1998

Mario Lorenz: "Geordnete Wege - Traffic Control mit Linux", iX 4/2000, Verlag Heinz Heise GmbH &Co KG

Internet Protocol - Quality of Service Page
http://qos.ittc.ukans.edu/

RFC 2549 "IP over Avian Carriers with Quality of Service"
http://www.ietf.org/rfc/rfc2549.txt

4.1.2. Grundlegende Verfahren

Dieser Abschnitt soll einen Überblick über existierende Verfahren bieten, von denen aber nur einige wirklich für die Versuche benötigt werden.

Admission Control

Admission Control legt fest, welcher Traffic im Netz überhaupt erlaubt ist. Festgelegt wird das über Policies. Es gibt zwei Ausprägungen: die passive Admission Control, in der es den Endnutzern überlassen wird, die Policies festzulegen, und die aktive Admission Control, in der das im **Ingress-Router** (d.h. dem Router, der die Daten von einem LAN in ein größeres Netz einspeist), realisiert wird.

Traffic Shaping

Traffic Shaping hat die Aufgabe, die Menge und Rate des Traffics zu kontrollieren, der in das Netz eintritt. Ziel dabei ist es, zumindest einen Anschein von vorhersagbarem Verhalten zu erhalten.

Leaky Bucket:

- Traffic-Bursts werden zwischengepuffert und in konstanten Raten wieder abgegeben
- wenn der Puffer voll ist, kommt es zu einem Überlauf und neu ankommende Pakete werden weggeschmissen
- Implementierung in der IP- oder z.B. in der ATM-Schicht
- Nachteil: wenn genug Bandbreite vorhanden ist, wird der Traffic unnötig eingeschränkt

Token Bucket:

- der Bucket wird hier statt mit Daten vom System in bestimmten Abständen mit Tokens versorgt
- jedes Token repräsentiert eine gewisse Menge von Datenbytes (festlegbar)
- Daten können nur gesendet werden (auch in Bursts), wenn genügend Tokens im Bucket sind
- zusätzlich ist eine maximale Burst-Größe festlegbar

Eine Kombination aus Token Bucket und Leaky Bucket wird verwendet, um zu verhindern, dass ein Traffic-Flow die ganze Bandbreite für sich allein beansprucht.

Preferential Queuing

Priority Queuing:

- höherpriorisierte Pakete werden vor niedrigpriorisierteren in die Output-Queue eingeordnet
- langsamer im Vergleich zum FIFO Queuing (da jedes Paket analysiert und ggf. umgeordnet werden muss)
- im schlimmsten Fall kommen niedrigpriorisierte Pakete nie bzw. "zu spät" an

Class-Based Queuing (CBQ):

- eine Variante von Priority Queuing mit mehreren Output-Queues
- Bandbreite wird hierarchisch (z.B. als Baum) auf Klassen je nach Bedarf verteilt
- Fairness: jede Klasse wird behandelt, auch niedrigpriorisierte Pakete kommen "pünktlich" an
- in der Praxis wird das durch die unterschiedliche Vergabe von Ressourcen geregelt
- verringerte Latenz gegenüber Priority Queuing, skaliert aber auch noch nicht gut

Weighted Fair Queuing (WFQ):

- Traffic mit niedriger Datenmenge wird bevorzugt behandelt
- Traffic mit höherer Datenmenge kann nicht sämtliche Ressourcen für sich beanspruchen
- jeder Traffic-Flow wird nach Datenmenge in die entsprechende Queue eingeordnet
- skaliert ebenfalls nicht (Rechenoverhead), außerdem zu statisch (wenig beeinflussbar)

Stochastic Fair Queuing (SFQ):

- Datenströme werden durch eine Hashbildung über Quell- und Zieladresse auf mehrere Queues verteilt, die gleichmäßig entleert werden (Round Robin)
- die Hash-Funktion wird in bestimmten Zeitintervallen geändert, um Hash-Kollisionen (verschiedene Verbindungen teilen sich dieselbe Warteschlange) zu minimieren
- einfacher als WFQ, aber nicht so leistungsfähig
- optimal in Verbindung mit CBQ einsetzbar

Clark-Shenker-Zhang (CSZ):

- ähnlich CBQ, aber präziser, dafür aber auch weniger flexibel und weniger effizient
- stellt wirklich garantierte Services zur Verfügung (garantierte Verzögerungen und Jitter)
- für jeden garantierten Service werden WFQ-Ströme erzeugt
- Rest der Bandbreite wird Dummystrom zugewiesen

Selective Forwarding

Das Grundprinzip des **Selective Forwarding** ist es, wichtige Daten auf einem schnellen Pfad zu befördern, unwichtigere Daten jedoch auf einem langsameren. Eine wichtige Rolle dabei spielen eine deterministische Pfadauswahl mit geringster Latenz, Jitter-Kontrolle u.a. Dazu gibt es eine Reihe von Verfahren (die aber selten eingesetzt werden):

TOS Routing:

- Routing in Abhängigkeit vom TOS-Byte:
- Bit 0-2: Routine, Priority, Immediate, Flash, Flash Override, CRITIC/ECP, Internetwork Control, Network Control
- Bit 3: Normal/Low Delay
- Bit 4: Normal/High Throughput
- Bit 5: Normal/High Reliability

Routing Information Protocol (RIP) und **Open Shortest Path First (OSPF)**:

- OSPF: kürzeste Wege nach Dijkstra (mit QoS im TOS-Byte)
- kompliziert im Vergleich zum zielbasierten Forwarding
- Ausweg: Policy-Based Routing (abhängig von Quelle statt Ziel), aber zu langsam

QoS Routing (QoSR):

- Absicht, QoS direkt in das Routing zu integrieren
- Probleme: asymmetrisches Routing (Hin- und Rückweg unterschiedlich), Overhead

Multi-Protocol Label Switching (MPLS):

- wird auch als Layer 2.5 bezeichnet (zwischen den Layern 2 und 3)
- das Routing erfolgt basierend auf mitgeführten Labels (20 Bit)
- bei ATM werden die Labels in VP/VC umgesetzt
- IP Precedence Bits als CoS in Label aufgenommen (3 Bit)

TCP-Mechanismen

TCP hat die Eigenschaft, in einer Überlastsituation, d.h. wenn Pakete des Datenstroms verloren gehen, die Datenrate herunterzufahren, sprich: die Pakete langsamer zu senden. Es existieren zwei Modi: der Slow-Start-Modus, bei dem die Datenrate nach dem Herunterfahren so weit wie möglich verdoppelt wird, und den Congestion-Avoidance-Modus, bei dem das Ansteigen linear erfolgt. Aber was passiert nun, wenn viele TCP-Ströme auf der Leitung fließen? Alle entdecken in etwa gleichzeitig eine Überlastsituation und alle gehen gleichzeitig in den Slow-Start-Modus. Während dieser Zeit ist die Leitung so gut wie frei (wenn man von anderen Protokollen absieht), es entsteht also ein Leerlauf. Die TCP-Ströme verdoppeln nun ihre Rate so lange, bis sie wieder eine Überlastsituation feststellen usw. Um dieser Fluktuation vorzubeugen, wurden folgende Verfahren entwickelt:

Random Early Detection/Drop (RED):

- verhindert den o.a. Überlastkollaps durch zufälliges Wegwerfen von Paketen (abhängig vom Füllstand der Queue)
- die Wahrscheinlichkeit eines Wegwerfens (Drops) ist festlegbar
- Ziel: Vermeidung der Situation, dass alle TCP-Flows gleichzeitig Überlast entdecken und in den Slow-Start-Modus gehen
- Effekt: TCP-Flows werden zu verschiedenen Zeitpunkten langsamer, Leerlauf und evtl. sogar eine echte Überlastung werden verhindert
- ist sehr effizient und fair, d.h. alle Daten werden gleich behandelt (keine CoS)
- Problem: UDP-Daten werden nicht erfasst, kommen aber im Multimedia-Bereich sehr oft vor

Weighted RED (WRED):

- auch als **Guaranteed Rate I/O (GRIO)** bezeichnet
- je höher die Precedence, desto geringer ist die Wahrscheinlichkeit eines Drops
- aktive Admission Control: z.B. über Token-Bucket-Schwellen (Tresholds)
- wenn Schwelle überschritten ->niedrigere Precedence (Treshold Triggering)

Generalized RED (GRED):

- mehrere Drop-Precedences festlegbar
- mehrere Drop-Wahrscheinlichkeiten, jeder Queue zugeordnet

Integrated Services (IntServ)

Integrated Services (IntServ) besitzt drei primäre Ziele:

1. die Services genau zu definieren
2. Application Service (End-To-End Ansprüche), Router Scheduling (welche Informationen sollen den einzelnen Routern verfügbar gemacht werden) und Link-Layer Interfaces ("Subnetze") zu definieren
3. Router Validation zu entwickeln, um sicherzustellen, dass Services auch zur Verfügung gestellt werden ->keine erhöhten zusätzlichen Anforderungen an Router

QoS tritt hier in einer ähnlichen Bedeutung auf wie bei ATM (erreichte Bandbreite, Paketverzögerung, Paketverlustraten usw.), findet aber im Gegensatz zu ATM in OSI-Layer 3 statt in Layer 2 statt. Es existieren 5 Schlüsselkomponenten:

1. QoS-Anforderungen: Serviceklassen und Traffic Control (Packet Scheduler, Classifier, Admission Control, Resource Reservation)
2. Resource-Sharing-Anforderungen: Link-Sharings, Weighted Fair Queuing (WFQ)
3. Erlaubnis, bestimmte Pakete wegzuschmeißen (Pakete, die nicht der Admission Control unterliegen oder durch Flags)
4. Festlegungen für Usage Feedback (Accounting Data)
5. **Resource Reservation Protocol (RSVP)**: dynamische QoS-Anforderungen

Differentiated Services (DiffServ)

Bei den **Differentiated Services (DiffServ)** existieren verschiedene **Queuing Disciplines (QDisc)** (z.B. TBF und RED), die an Serviceklassen gebunden werden. Die Klassen sind in einer Baumstruktur abgelegt, wobei 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. Den Blättern im Baum kann wieder eine QDisc zugewiesen werden, was DiffServ ziemlich leistungsfähig macht, denn so können verschiedene CoS durch unterschiedliche Algorithmen behandelt werden. Die Klassifikation erfolgt über Filter: generische Filter (z.B. routingbasiert) oder spezielle Filter (z.B. RSVP- und U32-Classifier).

Die Festlegung der Parameter geschieht meist nur in den Endpunkten oder beim Provider. Um die Klassifizierung der Flows mitzuführen, wird das **TOS-Byte** verwendet, was hier allerdings als **Differentiated Services Code Point (DSCP)** bezeichnet wird. In

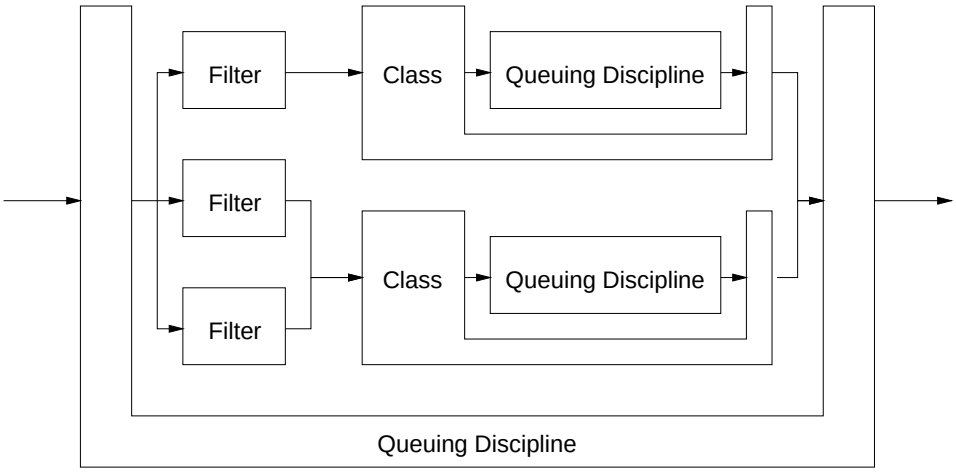

Abbildung 4.1-1.: Queuing Discipline mit mehreren Klassen

den Routern wird dann meist nur noch der DSCP ausgewertet, die Daten entsprechend behandelt und die Flows am Ende evtl. neu markiert, d.h. der DSCP angepasst. Linux unterstützt die Behandlung von DSCP's erst ab dem Kernel 2.3.x, weswegen sie für die Versuche noch keine Rolle spielen.

Die DSCP-Werte stammen aus drei Pools: Pool 1 (xxxxx0) ist für die Standard-Aktionen reserviert, Pool 2 (xxxx11) und Pool 3 (xxxx01) sind zu experimentellen Zwecken oder lokalem Gebrauch gedacht, wobei aber Pool 3 auch irgendwann zur Ergänzung von Pool 1 herangezogen werden kann. Der DSCP-Wert "xxx000" dient als Class-Selector CP, der Standard-CP ist "000000".

Man unterscheidet drei grundlegende Serviceklassen:

- **Assured Forwarding (AF)**: gesicherte Verbindungen, die in 4 verschiedene Klassen eingeteilt werden, von denen jede in unterschiedlichen Dropwahrscheinlichkeiten (z.B. über RED) vorkommt; z.B. für bevorzugte Kunden

	AF-Klasse 1	**AF-Klasse 2**	**AF-Klasse 3**	**AF-Klasse 4**
Niedrige Dropwkt.	001010 (0x0a)	010010 (0x12)	011010 (0x1a)	100010 (0x22)
Mittlere Dropwkt.	001100 (0x0c)	010100 (0x14)	011100 (0x1c)	100100 (0x24)
Hohe Dropwkt.	001110 (0x0e)	010110 (0x16)	011110 (0x1e)	100110 (0x26)

- **Expedited Forwarding (EF)**: ein so genannter Premium Service; z.B. für normale Kunden, mit DSCP = 101100 (0x2e)
- **Best Effort (BE)** bzw. **Default (DE)**: "herkömmliche" Verbindungen, in denen sich unwichtigerer Traffic die verbliebene Bandbreite nach dem Best-Effort-Prinzip teilt

Vergleicht man IntServ und DiffServ, stellt man fest, dass DiffServ wesentlich besser skaliert, was vor allem daran liegt, dass die Klassifikation der Datenströme an den Netzwerkrändern vorgenommen wird (in der Regel vom Provider) und in den Routern nur noch der DSCP ausgewertet werden muss. Als Nachteil ergibt sich daraus natürlich die fehlende Dynamik von DiffServ, da das Setup statisch erfolgt, wohingegen bei IntServ über RSVP eine dynamische Anpassung möglich ist.

Vertiefung:

References on CBQ (Class-Based Queueing)
http://www.aciri.org/floyd/cbq.html

References on RED (Random Early Detection) Queue Management
http://www.aciri.org/floyd/red.html

Differentiated Services on Linux
http://icawww1.epfl.ch/linux-diffserv/

IP Quality of Service: An Overview
http://www.ittc.ukans.edu/~rsarav/ipqos/ip_qos.htm

A Study of IP QoS
http://www.ittc.ukans.edu/~iyer/Mainpage.html

QoS Routing (qosr) - Charter
http://www.ietf.org/html.charters/qosr-charter.html

Integrated Services (intserv) - Charter
http://www.ietf.org/html.charters/intserv-charter.html

Resource Reservation Setup Protocol (rsvp) - Charter
http://www.ietf.org/html.charters/rsvp-charter.html

Differentiated Services (diffserv) - Charter
http://www.ietf.org/html.charters/diffserv-charter.html

RFC 1633 "Integrated Services in the Internet Architecture: An Overview"
http://www.ietf.org/rfc/rfc1633.txt

RFC 2205 "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification"
http://www.ietf.org/rfc/rfc2205.txt

RFC 2212 "Specification of Guaranteed Quality of Service"
http://www.ietf.org/rfc/rfc2212.txt

RFC 2474 "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers"
http://www.ietf.org/rfc/rfc2474.txt

RFC 2676 "QoS Routing Mechanisms and OSPF Extensions"
http://www.ietf.org/rfc/rfc2676.txt

IETF Internet Draft "A Framework for Multiprotocol Label Switching"
http://www.ietf.org/internet-drafts/draft-ietf-mpls-framework-05.txt

4.1.3. Realisierung in Linux

Der Traffic Shaper

Seit dem 2.0-er Kern existiert der Traffic Shaper, mit dem ein zusätzliches Device angelegt werden konnte, dessen Bandbreite dann explizit beschränkt wurde. Um den Netzverkehr über dieses Device zu leiten, mussten nur noch die Routen entsprechend gesetzt werden:

```
shapecfg attach shaper0 DEVICE
shapecfg speed shaper0 BANDWIDTH
ifconfig shaper0 HOST netmask NETMASK broadcast BROADCAST up
route add -net NETWORK netmask NETMASK dev shaper0
```

Dieses Verfahren war sehr einfach zu handhaben, aber auch sehr beschränkt in seiner Leistungsfähigkeit, so dass ab den späten 2.1-er Kernels *QoS and/or fair queueing* in den Kern integriert wurde und der Traffic Shaper kaum noch benutzt wird.

TC - Traffic Control

Im Paket IPRoute2 ist ein Tool zur Verwaltung von Queuing Disciplines, Klassen und Filtern enthalten: **Traffic Control (TC)**. Um neben der Benutzung dieser Applikation auch von eigenen Programmen auf QoS zurückgreifen zu können, wird eine API entwickelt. Generell muss QoS-Support im Kernel eincompiliert oder als Modul verfügbar sein (*Networking Options —>QoS and/or fair queueing*). Die benötigten Algorithmen sind ebenfalls als Modul oder in den Kernel eincompiliert auszuwählen. Weiterhin gibt es noch die Optionen *QoS/Rate Estimator*, der für QDiscs (z.B. CBQ) nötig ist, die die benutzte Bandbreite von Datenströmen abschätzen müssen, und die verschiedenen Classifier, die ausgewählt werden müssen, wenn die Klassifizierung nicht nur über das TOS-Byte erfolgen soll. Wenn man die QDiscs oder die Classifier als Module compiliert hat, sind sie explizit mit `modprobe sch_`*`qdisc`* bzw. `modprobe cls_`*`classifier`* zu laden, bevor sie verwendet werden können.

Nun soll die allgemeine Vorgehensweise zum Einrichten von QoS mittels TC erläutert werden: Zuerst muss an das gewünschte Netzinterface eine Queuing Discipline gebunden werden, nach der der Traffic (nur der ausgehende!) statt dem Einordnen in eine Standard-FIFO behandelt werden soll. Falls es sich bei dieser QDisc um CBQ (oder auch DSMark und ATM) handelt, sind nun noch Serviceklassen und die zugehörigen Classifier (Filter) zu definieren. Für die anderen QDiscs entfallen diese Schritte, hier wird der gesamte Traffic, der das Interface verlässt, nach der zugewiesenen Queuing Discipline behandelt. Im Falle von CBQ sind den Klassen (in der Regel den Blättern des Klassenbaums) wieder QDiscs zuzuweisen, nach denen der Traffic dieser Klasse behandelt werden soll. Unterbleibt dieser Schritt, wird wieder eine Standard-FIFO dafür

benutzt. Mit den Filtern wird durch eine Auswertung bestimmter Paketinformationen festgelegt, welche Traffic-Arten welchen Klassen zugeordnet werden.

> Hinweis!
> Für die Versuche steht eine vereinfachte Variante von TC zur Verfügung. Diese Übersicht ist hauptsächlich als Kommandoreferenz für weitergehende Anwendungen von QoS zu verstehen.

Zuweisung der QDisc

Mit folgendem Aufruf von TC wird eine QDisc einem Interface zugewiesen:

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

  dev         - Netzdevice
  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 | csz | dsmark | [p|b]fifo | prio | [g]red |
                 sfq | tbf | teql }
```

Folgende QDiscs sind implementiert:

- CBQ (Class-Based Queuing):

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

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

- CSZ (Clark-Shenker-Zhang): als defekt gekennzeichnet, wird wahrscheinlich ersetzt

- DSMark (Differentiated Services Field Marker, ab Kernel 2.3.x): ruft Classifier und speichert zurückgelieferte ClassID als TC-Index, sonst den Default-Index

```
tc qdisc add ... dsmark indices NUMBER [ default NUMBER ]
                 [ set_tc_index ]

  indices      - Anzahl Einträge in der Maske-Wert-Tabelle
  default      - Default-Index
  set_tc_index - DS-Byte soll als TC-Index gespeichert werden
```

- PFIFO (Packet FIFO) und BFIFO (Byte FIFO):

```
tc qdisc add ... [p|b]fifo limit NUMBER

  limit - Größe der Queue
```

- PRIO (Priority): bis zu 16 Unterwarteschlangen ("Bänder") mit unterschiedlicher Priorität (Band 0 die höchste); Default-Scheduler von Linux nutzt 3 Bänder

```
tc qdisc add ... prio bands NUMBER priomap P1 P2

  bands   - Anzahl der Bänder
  priomap - für Zuordnung der Pakete in die Bänder
```

- RED (Random Early Detection) und GRED (Generalized RED, ab Kernel 2.3.x):

```
tc qdisc add ... red limit BYTES min BYTES max BYTES avpkt BYTES
                 burst PACKETS probability PROB bandwidth KBPS
tc qdisc add ... gred setup DPs NUMBER default NUMBER [ grio ]
tc qdisc add ... gred limit BYTES min BYTES max BYTES avpkt BYTES
                 burst PACKETS probability PROB bandwidth KBPS
                 DP NUMBER prio NUMBER

  min/max     - minimale/maximale Paketgröße
  burst       - erlaubte Anzahl von Paketen in einem Burst
  probability - Drop-Wahrscheinlichkeit (Default: 2%)
  DPs         - Anzahl Drop-Precedences
  default     - Default Drop-Precedence
  grio        - GRIO (WRED) Buffer-Sharing
  DP          - Zuordnung zu einer Drop-Precedence
  prio        - Priorität
```

- SFQ (Stochastic Fair Queuing): maximal 128 Warteschlangen

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

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

- TBF (Token Bucket Filter):

```
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
```

- TEQL (True Link Equalizer): Daten über ein logisches Interface (teqlN) auf mehrere physikalische Interfaces verteilt (je nach Bandbreite der Interfaces)

```
modprobe sch_teql
tc qdisc add dev DEVICE_1 root TEQL_DEV
 :
tc qdisc add dev DEVICE_N root TEQL_DEV
ifconfig TEQL_DEV HOST [ netmask NETMASK broadcast BROADCAST ] up
route add ... TEQL_DEV
```

Ab dem Kernel 2.3.x existieren auch noch eine Ingress QDisc für ankommenden Traffic und eine ATM VC Selection QDisc.

Klassenhierarchie aufbauen

Falls als QDisc CBQ, DSMark oder ATM verwendet wurde, werden im zweiten Schritt die Klassen für die verschiedenen Traffic-Arten eingerichtet und in einem Baum angeordnet, indem beim Anlegen einer Klasse immer der Parent-Knoten mit angegeben wird. Innherhalb des CBQ-Klassenbaums kann wie bei DiffServ Bandbreite zwischen den einzelnen Klassen verborgt werden. Die Ratenangaben beim Anlegen der Klassen dienen hierbei nur zur Aufteilung der Bandbreite, eine explizite Begrenzung muss durch TBF erfolgen.

```
tc class add dev DEVICE classid CLASSID { root | parent CLASSID }
             QDISC_KIND OPTIONS

  classid    - ID der Form X:Y (X = ID der QDisc, Y = ID der Klasse),
               Root-Klasse hat X:0, andere z.B. X:1, X:2, ...
  QDISC_KIND = { atm | cbq | dsmark } (andere QDiscs sind klassenlos)
```

- CBQ (Class-Based Queuing): siehe auch DiffServ im vorigen Abschnitt

```
tc class add ... cbq bandwidth BPS rate BPS [ avpkt BYTES ]
                 [ mpu BYTES ] [ allot BYTES ] [ weight RATE ]
                 maxburst PACKETS [ minburst PACKETS ]
                 [ prio NUMBER ] [ cell BYTES ] [ ewma LOG ]
                 [ bounded ] [ isolated ]
                 [ estimator INTERVAL TIME_CONSTANT ]
                 [ split CLASSID ] [ defmap MASK/CHANGE ]

  rate      - zugewiesene Bandbreite
  maxburst  - maximale Anzahl von Paketen in einem Burst
  minburst  - minimale Anzahl von Paketen in einem Burst
  bounded   - Klasse darf sich keine Bandbreite borgen
  isolated  - Klasse teilt sich keine Bandbreite mit Nicht-Kindern
  allot     - MTU + Größe des MAC-Headers
  weight    - Gewicht der Klasse (optional, proportional zu "rate")
  prio      - Priorität der Klasse (zwischen 1 und 8, 1 = höchste)
  cell      - Anzahl Bytes, in der Transferzeit gemessen wird
  split     - eingesetzter Classifier gilt nur für Klassen mit
              entsprechendem Split-Wert
  defmap    - bei ungematchten Paketen wird TOS-Byte mit Maske
              ausgewertet
```

- DSMark (Differentiated Services Field Marker, ab Kernel 2.3.x): ändert den DSCP

```
tc class change ... dsmark mask MASK value VALUE

  mask + value - Berechnung des DSCP: DSCP = (DSCP &  MASK) | VALUE
```

Klassifizierung der Pakete

Schlussendlich müssen nur noch die einzelnen Pakete ihren Klassen (auch als Flows bezeichnet) zugeordnet werden, indem bestimmte Informationen in den Paketen ausgewertet werden. Die Filterregeln werden der Reihe nach durchgegangen, bis eine der Regeln zutrifft, so dass der Traffic einer Klasse zugeordnet werden kann. Ansonsten wird er als Best-Effort-Traffic behandelt.

```
tc filter add dev DEVICE [ handle FILTERID ] { root | parent CLASSID }
              [ prio PRIO ] [ protocol PROTO ]
              [ estimator INTERVAL TIME_CONSTANT ] FILTER_TYPE OPTIONS

  parent      - Handle der QDisc
  handle      - Handle des Filters (siehe die einzelnen Classifier)
  prio/pref   - Priorität des Filters
  PROTO       = { ip | ... }
  FILTER_TYPE = { route | fw | u32 | rsvp[6] | tcindex }
```

Folgende Klassifizierungsmöglichkeiten stehen zur Auswahl:

- über das Routing-Subsystem: Mit `ip route` wird eine Route gesetzt und dieser eine Nummer zugewiesen, die dann vom Filter an einen Flow, d.h. an ein Blatt im Klassenbaum, gebunden wird.

  ```
  ip route add NET_ADDRESS [ via GATEWAY ] dev DEVICE realms REALM
  tc filter add...route [ from REALM | fromdev DEVICE | fromif TAG ]
                  [ to REALM ] [ flowid CLASSID ] [ police POLICE ]

    flowid - Blatt im Klassenbaum
    POLICE = rate BPS buffer BYTES[/BYTES] burst BYTES[/BYTES]
    (s. TBF) [ mpu BYTES[/BYTES] ] [ mtu BYTES[/BYTES] ]
             [ peakrate BPS ] [ avrate BPS ] [ index NUMBER ]
             [ ACTION ]
    ACTION = { reclassify | drop | continue }
  ```

- über das Firewall-Subsystem: Mit `ipchains` wird eine Firewallregel erzeugt und mit einer Nummer markiert, die der Filter dann wieder an den Flow binden kann (über das Handle!).

  ```
  ipchains -A output -s NET_ADDRESS -m NUMBER
  tc filter add ... handle NUMBER fw [ flowid CLASSID ]
                   [ police POLICE ]
  ```

- über den U32-Classifier: Mit den Match-Anweisungen wird der Paketkopf ausgewertet, und durch die FlowID geschieht die Zuordnung zu einer Klasse. Ein Match-Statement 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).

```
tc filter add ... [ handle X:Y:Z ] u32 [ match SELECTOR ]
                  flowid CLASSID [ offset mask MASK shift NUMBER ]
                  [ hashkey mask MASK at NUMBER ] [ link HTID ]
                  [ ht HTID ] [ police POLICE ] [ sample SAMPLE ]
tc filter add ... handle X: u32 divisor DIVISOR

  divisor  - Hashtable mit DIVISOR Slots und Handle X: erzeugen
  ht       - Zuordnung zu Y-tem Slot der Hashtable X (HTID=X:Y:)
  offset   - Offset in der Hashtable
  hashkey  - Key in der Hashtable
  SELECTOR = SAMPLE SAMPLE ...
  SAMPLE   = { ip | ip6 | udp | tcp | icmp | u{8|16|32} }
             FIELD [ TYPE MASK ]
```

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

- über den RSVP-Classifier: Hier kann die Bandbreite dynamisch mittels RSVP je nach Anforderung vergeben werden (Bandwidth on Demand):

```
tc filter add ... handle X:Y rsvp ipproto PROTOCOL session
                  DST[/PORT | GPI ] [ sender SRC[/PORT | GPI ] ]
                  [ classid CLASSID ] [ police POLICE ]
                  [ tunnelid ID ] [ tunnel ID skip NUMBER ]

  ipproto - Protokoll
  GPI     - Generalized Port Identifier:
            { flowlabel NUMBER | spi/ah SPI | spi/esp SPI |
              u{8|16|32} NUMBER mask MASK at OFFSET }
  SPI     - Source Port ID
```

- über den TC-Index-Classifier (ab Kernel 2.3.x):

```
tc filter add ... [ handle DSCP ] tcindex [ classid CLASSID ]
                  [ mask MASK ] [ shift NUMBER ]
                  [ pass_on ] [ fall_through ]

  mask + shift - Keyberechnung: KEY = (TC-Index » NUMBER) &  MASK
  pass_on      - wenn handle != DSCP, dann nächsten Filter suchen
  fall_through - versucht, eine neue Klasse zu erzeugen, wenn es
                 keine zu KEY passende gibt
```

Eingerichtete QoS anzeigen

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

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

Vertiefung:

Linux - Advanced Networking Overview
http://qos.ittc.ukans.edu/howto/

Linux 2.4 Advanced Routing HOWTO
http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html

Linux QoS Support
http://www.ittc.ukans.edu/~rsarav/projects/networking/ipqos/diffoverview/

Differentiated Services on Linux
ftp://lrcftp.epfl.ch/pub/linux/diffserv/misc/dsid-01.txt

IP QoS Efforts
http://qos.ittc.ukans.edu/slides/

An API for Linux QoS Support
http://www.ittc.ukans.edu/~pramodh/courses/linux_qos/mainpage.html

Linux Iproute2, Traffic Control &Friends
http://defiant.coinet.com/iproute2/

README zu IPRoute2 und TC
file:///usr/doc/packages/iproute/README.iproute2+tc

Traffic Shaper For Linux
file:///usr/src/linux/Documentation/networking/shaper.txt

Linux-Kernelquellen
file:///usr/src/linux/net/sched/

4.2. Versuchsumgebung

4.2.1. Versuchsaufbau

Für die Versuche stehen vier Rechner zur Verfügung, die drei miteinander verbundene Netzwerke bilden. Der Rechner *athos* fungiert als Router, der die Netze verbindet.

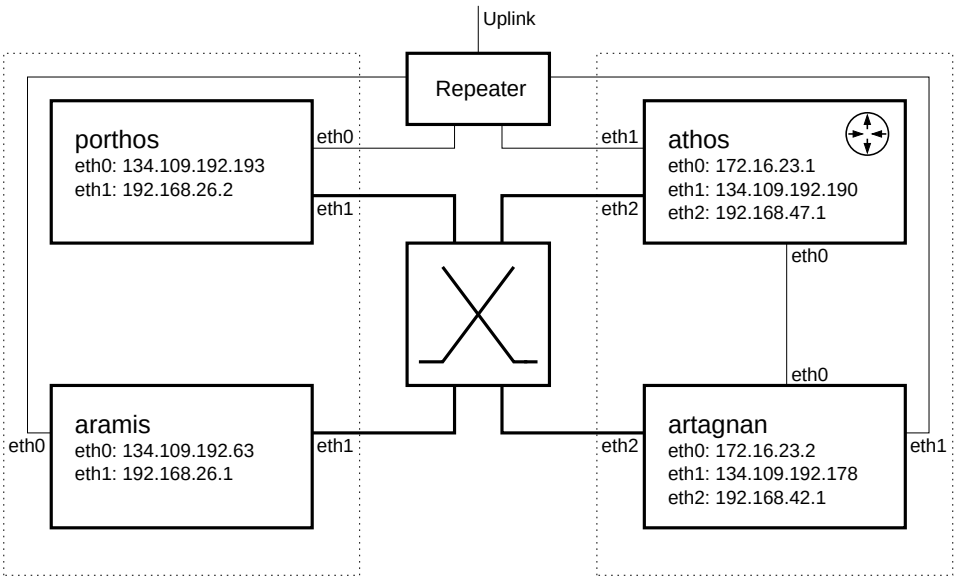

Abbildung 4.2-1.: Versuchsaufbau

Gigabit-Netz

Wie aus dem Gigabit-Versuch noch bekannt sein sollte, besteht das Gigabit-Netz aus drei VLAN's: *aramis* und *porthos* mit der Netzadresse 192.168.26/24, *artagnan* mit 192.168.42/24 und *athos* mit 192.168.47/24. Als Default-Route ist der Switch mit 192.168.xx.254 eingetragen, wobei xx die Nummer des Subnetzes darstellt. Der Switch ist so konfiguriert, dass er Pakete defaultmäßig an *athos* schickt, wenn sie eine unbekannte Zieladresse enthalten.Für diesen Versuch spielt das Gigabit-Netz zwar keine Rolle, aber der Aufbau ist dennoch hier enthalten, um die Routingeinstellungen verstehen zu können.

100MBit-Netz

Alle Rechner besitzen auch eine offizielle IP aus dem 134.109.192/24-er Netz der Informatik und sind für den Versuch mit einem 10/100MBit-Repeater verbunden. Der besitzt zwar einen Uplink nach außen, dieser sollte aber bei der Versuchsdurchführung gekappt werden, um nicht das gesamte Subnetz zu beeinträchtigen.

100MBit-Direktverbindung

Zusätzlich existiert noch eine 100MBit-Direktverbindung zwischen *athos* und *artagnan* mit der Netzadresse 172.16.23/24.

Hinweis!
Da das Labor hauptsächlich für das *Gigabit Ethernet Praktikum* gedacht ist, müssen vor Beginn der Versuche noch einige Einstellungen vorgenommen werden: Auf allen Rechnern ist die Default-Route zu löschen und auf *athos* zu setzen, d.h. `route add default gw 172.16.23.1 eth0` für *artagnan* und `route add default gw 134.109.192.190 eth0` bei den beiden anderen Rechnern. Zusätzlich ist auf *artagnan* noch die Route ins 134.109.192/24-er Netz zu löschen, damit die Pakete auf dem Rückweg auch über den Router laufen. Der Aufruf von `routes set` auf einem der Rechner erledigt das für Sie. Um mit den Rechnern arbeiten zu können, benötigen Sie natürlich noch das Root-Passwort, es lautet: `Praktikum`. Zwischen den Rechnern steht Ihnen allerdings eine passwortlose SSH zur Verfügung.

4.2.2. Zur Verfügung stehende Software

Da die Handhabung des TC-Kommandos sehr komplex ist, gibt es für diesen Versuch eine vereinfachte Variante namens **Traffic Control Changer (TCC)**, der auch gleich die benötigten Module mit lädt und folgende Syntax besitzt:

```
tcc cbq    DEVICE HANDLE { root | PARENTID } BANDWIDTH
tcc red    DEVICE HANDLE { root | PARENTID } BANDWIDTH PROBABILITY
tcc sfq    DEVICE HANDLE { root | PARENTID }
tcc tbf    DEVICE HANDLE { root | PARENTID } RATE
tcc teql   DEVICE
tcc class  DEVICE CLASSID PARENTID BANDWIDTH RATE WEIGHT PRIO \
           [ bounded ] [ isolated ]
tcc filter DEVICE FLOWID PARENTID match SELECTOR
tcc remove DEVICE
tcc list   DEVICE [ -s ]
```

- Die ersten vier Aufrufe binden eine Queuing Discipline an ein Interface (`root`) bzw. an ein Blatt im Klassenbaum mit der ClassID PARENTID. Bei TEQL muss nur das Netzdevice angegeben werden.
- Durch `tcc class` wird eine Klasse mit der ClassID CLASSID und der Elternklasse PARENTID angelegt; handelt es sich um die oberste Klasse, ist als PARENTID X:0 anzugeben, wobei X: das Handle der QDisc darstellt.
- BANDWIDTH ist immer die reale Bandbreite des Interfaces DEVICE, RATE die gewünschte (natürlich nur kleiner oder gleich BANDWIDTH) und WEIGHT das zu RATE proportionale Gewicht (z.B. 1/10 von RATE).
- Mit `tcc filter` wird über einen SELECTOR der Traffic, der das Interface verlässt, der Klasse mit der ClassID FLOWID zugeordnet, PARENTID ist in dem Fall das Handle der QDisc (also X:0).
- Die restlichen Parameter sollten nach der Lektüre der TC-Syntax selbsterklärend sein. Für die hier nicht angegebenen TC-Parameter werden durch den TCC Standardwerte gesetzt.

Hinweis!
Da die einzugebenden Kommandos immer noch recht umfangreich sind und auch leicht Fehler auftreten können, empfiehlt es sich, sie in einem Shell-Skript zu "verewigen", ehe man sie ausführt. Fügen Sie dem Protokoll bitte immer den Quelltext ihrer Skripte hinzu!

Ein einfaches Beispiel, um UDP-Traffic auf 1MBit zu begrenzen, könnte etwa so aussehen:

```
# QDisc erzeugen und an das Interface binden
tcc cbq    eth0 1:  root 100MBit

# Root-Klasse mit der ID 1:1 anlegen
tcc class  eth0 1:1 1:0  100MBit 100MBit   10MBit 8

# Klasse fuer Rest ohne QDisc und Filter
tcc class  eth0 1:2 1:1  100MBit  99MBit 9900KBit 5

# Klasse fuer UDP mit QDisc TBF und Filter (Protokoll UDP)
# erst durch TBF wird die Bandbreite endgueltig beschraenkt
tcc class  eth0 1:3 1:1  100MBit   1MBit  100KBit 5 bounded
tcc tbf    eth0 2:  1:3            1MBit
tcc filter eth0 1:3 1:0  match ip protocol 17 0xff
```

Wollte man statt UDP den gesamten ausgehenden Traffic eines bestimmten Rechners oder Netzes begrenzen (z.B. im Router), so muss man lediglich einen anderen Filter verwenden. In diesem Beispiel wird der gesamte Traffic des CSN-Subnetzes der V54 begrenzt.

```
tcc filter eth0 1:3 1:0  match ip src 134.109.96.0/22
```

Falls man diese Maßnahme im Router vornimmt, kann man durch eine analoge Vorgehensweise auf dem anderen Interface auch den eingehenden Traffic des Subnetzes begrenzen (in Wirklichkeit wird ja der in Richtung Subnetz ausgehende Traffic im Router begrenzt).

```
...
tcc filter eth1 3:3 3:0  match ip dst 134.109.96.0/22
```

Hinweis!
Zwischen den Versuchen ist auf allen benutzten Interfaces ein `tcc remove DEVICE` auszuführen und mit `tcc list DEVICE` zu überprüfen, ob auch alle QDiscs entfernt wurden. Falls das nicht der Fall sein sollte (z.B. durch Fehleingaben verursacht), ist es am einfachsten, den Rechner neu zu starten.

Um die vorgenommenen Änderungen zu messen bzw. anderweitig zu überprüfen, gibt es folgende Möglichkeiten:

FTP und SCP

Die beiden Programme zeigen die Übertragungsrate während des Dateitransfers oder danach an. Dateien zum Transfer liegen unter `/root/prak_files/`. Bei SCP können Sie mit der Option `-c blowfish` veranlassen, dass der etwas schnellere Verschlüsselungsalgorithmus *Blowfish* verwendet wird, was den Durchsatz merklich erhöht.

NetPIPE und GnuPlot

Mit NetPIPE wird eine TCP-Messreihe gestartet, bei der im Ping-Pong-Verfahren Datenblöcke gesendet werden, deren Größe stetig erhöht wird. Unter `/root/prak_messungen/` werden die Ergebnisse in Files abgelegt, die mit GnuPlot ausgewertet werden können. Um eine Messreihe zu starten, muss auf dem Empfängerrechner `netpipe recv` aufgerufen werden, und auf dem Senderrechner `netpipe send`. Bei mehreren Sendern sind auch entsprechend viele Receiver zu starten und anhand der Portnummer zu unterscheiden:

```
netpipe send HOST PORT MESSFILE
netpipe recv PORT
```

> Hinweis!
> Die NetPIPE-Messreihen nehmen relativ viel Zeit in Anspruch: ca. 5min bei 100MBit/s. Diese Zeit gilt für **jede** Messreihe; auch wenn mehrere parallel durchgeführt werden, addieren sich die Zeiten, da sich dann die einzelnen Datenströme behindern.

Zur grafischen Auswertung ist nach dem Starten von GnuPlot mittels `gnuplot` am dortigen Kommandoprompt folgendes einzugeben, um den Durchsatz (2. Spalte im File) der Messreihe über der Blockgröße (4. Spalte) abzutragen:

```
load "/root/praktikum/ds.gnu"
plot "MESSFILE1" using 4:2 w l,"MESSFILE2" using 4:2 w l, ...
```

TCP-Dump und Ethereal

Mit `tcpdump -i INTERFACE` können Sie den Traffic auf einem Netzwerkinterface überwachen. Durch die Option `-p` ist zu verhindern, dass die Karte in den Promiscous-Modus gesetzt wird, damit Sie wirklich nur den Traffic des betreffenden Rechners "erwischen". Falls Sie mit Grep bestimmte Informationen herausfiltern wollen, müssen Sie zusätzlich noch die Option `-l` angeben oder die eingebauten Filter benutzen. Das Tool Ethereal bietet eine grafische Oberfläche zur Netzwerkanalyse und -diagnose.

IfConfig

Damit können Sie u.a. feststellen, wieviele Pakete auf den einzelnen Interfaces empfangen und gesendet wurden, am besten in Verbindung mit Watch: `watch -n INTERVAL ifconfig`.

Logfiles

Falls Sie Einsicht in die Logfiles benötigen... diese liegen unter unter `/var/log/` und `/var/log/httpd/`.

Screenshots

Screenshots von grafischen Ausgaben können z.B. mit XV gemacht werden. Dazu einfach auf den Grab-Button klicken und das gewünschte Fenster auswählen.

4.3. Versuchsdurchführung

4.3.1. Versuch: Verschiedene Klassen mit CBQ

Szenario

Der Rechner *porthos* führt mittels `ping -f -s 8000 172.16.23.2` eine Flood-Ping-Attacke gegen *artagnan* durch. Von dem anderen Rechner versucht ein Nutzer mittels SCP ein File auf den angegriffenen Rechner zu kopieren.

Frage 4.3.1.1:
Was ist während der Attacke zu erwarten? Begründen Sie ihre Vermutungen und vergleichen Sie sie mit den praktischen Ergebnissen!

Die erreichte Übertragungsrate bricht extrem ein. Das liegt daran, dass die SCP-TCP-Verbindung gegen das "rücksichtslosere" ICMP-Protokoll keine Chance hat, sondern immer wieder in den Slow-Start-Modus gerät und wesentlich weniger bzw. gar keine Pakete mehr übertragen kann.

Aufgabe

Starten Sie 4 SCP-Filetransfers von *aramis* zu 172.16.23.2:

1. ohne QoS-Maßnahmen und ohne Attacke (der "Idealfall")
2. ohne QoS-Maßnahmen und mit Attacke (der "Angriffsfall")
3. mit QoS-Maßnahmen und ohne Attacke (um zu sehen, ob QoS zu Performance-Einbußen führt)
4. mit QoS-Maßnahmen und mit Attacke (der "Abwehrfall")

Begrenzen Sie dazu im Router die Bandbreite für ICMP und UDP mittels CBQ und TBF, indem Sie vom Beispiel im vorhergehenden Abschnitt ausgehen! Die Begrenzung soll dabei so ausgelegt sein, dass sich die Klassen für ICMP und UDP keine Bandbreite borgen dürfen. UDP wird hier zwar nicht genutzt, soll aber trotzdem ebenfalls begrenzt werden, da statt einem Flood-Ping z.B. auch eine bandbreitenintensive UDP-Anwendung denkbar wäre. Protokollnummern finden Sie in `/etc/protocols`. Falls Sie genügend Zeit haben, können Sie auch mit verschiedenen Paketgrößen bei `ping -f` experimentieren.

Frage 4.3.1.2:
Auf welchem Interface im Router müssen die QoS-Maßnahmen ergriffen werden? Begründen Sie die getroffene Wahl!

Auf dem Interface vom Router zum angegriffenen Rechner, da nur der ausgehende Traffic zum angegriffenen Rechner begrenzt werden kann.

Welche Transferraten haben Sie in den 4 Fällen erhalten? Erklären Sie die Ergebnisse!

4.3.2. Bemerkungen zum Versuch 1

Der erste Versuch soll demonstrieren, wie man explizit die Bandbreite einer bestimmten Art von Netztraffic beschränken kann. Gleichzeitig soll gezeigt werden, dass der bloße Einsatz von QoS kaum bis überhaupt nicht zu Performance-Einbußen führt. Dieser Versuch ist sehr einfach und deshalb meiner Meinung nach gut als Einstieg geeignet. Die Durchführung nimmt auch nur wenig Zeit in Anspruch.

> Hinweis!
> Beim Testen der Versuche habe ich sehr viele Messungen mit zum Teil unterschiedlichen Parametern vorgenommen, um möglichst genaue Resultate zu erhalten. Aus Zeitgründen ist es nicht zwingend notwendig, dass alle Messungen von den Studenten durchgeführt werden. Allerdings habe ich trotzdem darauf hingewiesen, was man noch ausprobieren könnte, wenn die Zeit reicht. Auch bei den Kontrollfragen gibt es mehrere mögliche Antworten.

Musterlösung

```
tcc cbq    eth0 1: root 100MBit
tcc class  eth0 1:1 1:0 100MBit 100MBit   10MBit 8
tcc class  eth0 1:2 1:1 100MBit  98MBit 9800KBit 5          # Rest
tcc class  eth0 1:3 1:1 100MBit   1MBit  100KBit 5 bounded  # ICMP
tcc class  eth0 1:4 1:1 100MBit   1MBit  100KBit 5 bounded  # UDP
tcc tbf    eth0 2:  1:3           1MBit                     # ICMP
tcc tbf    eth0 3:  1:4           1MBit                     # UDP
tcc filter eth0 1:3 1:0 match ip protocol  1 0xff           # ICMP
tcc filter eth0 1:4 1:0 match ip protocol 17 0xff           # UDP
```

Fall	QoS	Attacke	Ping-Paketgröße	SCP-Rate in KB/s
Idealfall	-	-	-	2639.4
Angriffsfall	-	X	8000	abgebrochen
			16000	abgebrochen
			64000	97.7
mit QoS	X	-	-	2639.4
Abwehrfall	X	X	8000	2639.4
			16000	2639.4
			64000	2569.9

Wie erwartet, kann man mit SCP nicht mehr arbeiten, sobald ein Flood-Ping stattfindet. Durch ein Begrenzen des ICMP-Traffics kann dem effektiv entgegengewirkt werden. Es ist auch kein Performance-Verlust feststellbar, wenn QoS eingesetzt wird.

4.3.3. Versuch: Borgen von Bandbreite mit CBQ

Szenario

Die beiden Rechner *aramis* und *porthos* sollen das CSN darstellen, *athos* fungiert als Gateway, und *artagnan* (172.16.23.2) ist ein Computer außerhalb des Uninetzes. Die Verantwortlichen des CSN planen nun, Bandbreitenbeschränkungen einzuführen. Der Traffic soll dabei in folgende Klassen aufgeteilt werden: SSH für das interaktive Arbeiten und eine Klasse für den restlichen Traffic (der dann u.a. FTP umfasst). Das Borgen von nicht benötigter Bandbreite soll möglich sein.

Frage 4.3.3.1:
Wie würden Sie die Bandbreite auf die Klassen aufteilen (prinzipiell)? Mit welchen Prioritäten? Begründen Sie!

Der SSH könnte man etwas weniger Bandbreite geben, da das interaktive Arbeiten in der Regel nicht soviel benötigt, während die Bandbreite für FTP-Transfers nötiger ist. Wenn FTP die Bandbreite nicht nutzt, kann sie ja verborgt werden. Außerdem sind FTP-Transfers nach bzw. von außen sicherlich häufiger als SCP-Transfers. Die SSH sollte allerdings eine höhere Priorität erhalten, damit ein interaktives Arbeiten mit geringen Antwortzeiten möglich ist.

Aufgabe

Setzen Sie im Router die geplanten Beschränkungen um! Um alle FTP-Nutzer gleich zu behandeln, soll bei der Rest-Klasse außerdem die QDisc SFQ zum Einsatz kommen. Da die Verbindungen in beide Richtungen begrenzt werden sollen, müssen die QoS-Maßnahmen auf beiden Interfaces des Routers ergriffen werden (da nur der jeweils ausgehende Traffic begrenzt werden kann). Für jedes Interface muss der SSH-Traffic in 2 Klassen unterteilt werden (SSH-Client und SSH-Daemon), wobei die Klassifizierung durch Auswertung von Quell- bzw. Zielport 22 erfolgen kann, während der Rest-Traffic z.B. durch die Netzadresse 134.109.192.0/24 (je nach Transferrichtung als Quell- oder Zieladresse) richtig zugeordnet werden kann. Denken Sie aber in dem Fall daran, die Filterregel der Rest-Klasse als letztes zu definieren, da sie sonst immer zutreffen würde, ehe die SSH-Regeln zum Zuge kommen könnten.

Frage 4.3.3.2:
Warum benutzt man, um den Rest-Traffic zu klassifizieren, die Netzadresse des "CSN's" und nicht die des außerhalb liegenden Netzes?

Weil die außerhalb liegenden Adressen in der Realität nur in den seltensten Fällen bekannt sein dürften, während die internen Adressen meist festgelegt sind.

Hinweis!
Wählen Sie als Bandbreite für die SSH-Klassen nur 10MBit, da die beiden Rechner es kaum schaffen, durch SCP mehr als diese auszulasten (wegen der Verschlüsselung). Durch ein `scp -c blowfish` kommen Sie nur auf unwesentlich mehr als 32MBit, und das ist der schnellste mögliche Algorithmus. Im realen CSN stehen natürlich wesentlich mehr Rechner zur Verfügung, so dass man dort auch eine höhere Bandbreite wählen kann, bei den Versuchen würden Sie aber sonst keine sinnvollen Resultate erhalten.

Führen Sie Datentransfers per SCP und/oder FTP von einem und von beiden "CSN"-Rechnern gleichzeitig durch, um ihre getroffenen Maßnahmen zu überprüfen. Lassen Sie dabei jede Klasse erst einmal einzeln "zu Wort kommen", um zu sehen, ob das Borgen funktioniert. Danach beanspruchen Sie bitte immer mindestens 3 Klassen, damit das Netz wenigstens annähernd ausgelastet ist. Ein interessanter Fall ist es z.B., die SSH-Klassen von *eth0* und die FTP-Klasse von *eth1* zu benutzen, testen Sie aber auch andere! Messen Sie die Transferraten und fassen Sie ihre Ergebnisse zusammen! Durch `tcc list INTERFACE -s` können Sie u.a. nachschauen, welche Klasse sich Bandbreite geborgt hat (Angabe hinter "borrowed").

4.3.4. Bemerkungen zum Versuch 2

Im zweiten Versuch soll das Borgen von Bandbreite zwischen verschiedenen Klassen demonstriert werden. Am interessantesten sind dabei die Varianten, bei denen verschiedene Filetransfermethoden (SCP und FTP) gleichzeitig in verschiedene Richtungen stattfinden und das Netz auch auslasten. Dieser Versuch ist schon etwas komplizierter, und auch seine Durchführung kann je nach Art und Anzahl der einzelnen Messungen wesentlich länger dauern.

Bei SCP ist zu beachten, dass ein solcher Datenstrom das Netz selbst bei Verwendung des schnelleren *Blowfish*-Verschlüsselungsalgorithmus' nicht auslasten kann. Ich habe mich trotzdem dafür entschieden, SCP zu verwenden, weil es zum einen in der Realität (hoffentlich) häufiger anzutreffen ist als RCP, und zum anderen, weil Root keinen RCP-Transfer durchführen darf, sondern extra ein Nutzer dafür angelegt werden müsste. Durch eine geeignete Bandbreiten-Begrenzung (siehe Versuchsanleitung) lassen sich trotzdem vernünftige Resultate erzielen.

Musterlösung

```
# Externes Interface
tcc cbq    eth0 1: root 100MBit
tcc class  eth0 1:1 1:0 100MBit 100MBit 10MBit 8
tcc class  eth0 1:2 1:1 100MBit  10MBit  1MBit 2  # SSH
tcc class  eth0 1:3 1:1 100MBit  10MBit  1MBit 2  # SSHD
tcc class  eth0 1:4 1:1 100MBit  80MBit  8MBit 5  # FTP+Rest
tcc sfq    eth0 2:  1:4

tcc filter eth0 1:2 1:0 match ip dport 22 0xffff
tcc filter eth0 1:3 1:0 match ip sport 22 0xffff
tcc filter eth0 1:4 1:0 match ip src 134.109.192.0/24

# Internes (CSN-)Interface
tcc cbq    eth1 3: root 100MBit
tcc class  eth1 3:1 3:0 100MBit 100MBit 10MBit 8
tcc class  eth1 3:2 3:1 100MBit  10MBit  1MBit 2  # SSH
tcc class  eth1 3:3 3:1 100MBit  10MBit  1MBit 2  # SSHD
tcc class  eth1 3:4 3:1 100MBit  80MBit  8MBit 5  # FTP+Rest
tcc sfq    eth1 4:  3:4

tcc filter eth1 3:2 3:0 match ip dport 22 0xffff
tcc filter eth1 3:3 3:0 match ip sport 22 0xffff
tcc filter eth1 3:4 3:0 match ip dst 134.109.192.0/24
```

Vers.	Quelle	Ziel	Initiator	Interf.	Klasse	Rate in MB/s	
						mit QoS	ohne QoS
1	aramis	artagnan	aramis	eth0	SSH	4.25	4.25
2	aramis	artagnan	aramis	eth0	FTP	10.24	9.68
3	aramis	artagnan	aramis	eth0	FTP	5.27	5.20
	porthos	artagnan	porthos	eth0	FTP	4.87	4.97
4	aramis	artagnan	aramis	eth0	SSH	1.71	1.81
	porthos	artagnan	porthos	eth0	FTP	7.47	7.76
5	aramis	artagnan	artagnan	eth0	SSHD	1.68	1.74
	porthos	artagnan	porthos	eth0	FTP	7.63	7.83
6	aramis	artagnan	aramis	eth0	SSH	1.02	0.97
	aramis	artagnan	artagnan	eth0	SSHD	0.94	0.96
	porthos	artagnan	porthos	eth0	FTP	7.66	6.78
7	artagnan	aramis	artagnan	eth1	SSH	4.07	-
8	artagnan	aramis	aramis	eth1	FTP	10.42	-
9	artagnan	aramis	artagnan	eth1	SSH	3.76	-
	artagnan	porthos	porthos	eth1	FTP	6.91	-
10	aramis	artagnan	aramis	eth0	SSH	2.21	-
	artagnan	porthos	artagnan	eth1	SSH	2.64	-
11	aramis	artagnan	aramis	eth0	FTP	5.17	-
	artagnan	porthos	porthos	eth1	FTP	5.27	-
12	aramis	artagnan	aramis	eth0	SSH	1.11	0.84
	aramis	artagnan	artagnan	eth0	SSHD	0.96	0.84
	artagnan	porthos	porthos	eth1	FTP	5.40	5.94

Der Initiator ist derjenige Rechner, von dem aus ein Transfer durchgeführt wird. Diese Unterscheidung ist wichtig für die Zuordnung des Traffics zur SSH- bzw. SSHD-Klasse. Bei FTP ist es egal, von welchem Rechner aus der Transfer initiiert wird, da hier alles zur "Rest"-Klasse zählt.

In den ersten drei Versuchen ist zu sehen, dass die Bandbreite tatsächlich verborgt wird. Die Tatsache, dass in Versuch 3 nur jeweils ca. 5MB/s Transferrate erreicht wird, liegt nur an der gegenseitigen Behinderung der Datenströme, wie man auch an den Raten ohne QoS sehen kann. In den Versuchen 4-6 sieht man, wie die Bandbreite verteilt wird, wobei sich in Versuch 4 und 5 das SCP bei der jeweils anderen SSH-Klasse noch Bandbreite borgt. Bei Versuch 6 sind alle drei Klassen vertreten und nutzen ihre zugewiesene Bandbreite (weitestgehend) aus, so dass dort nichts mehr verborgt werden kann. Hier ist sogar ein höherer Gesamtdurchsatz gegenüber desselben Versuchs ohne QoS zu verzeichnen, wohl wegen der verminderten Kollisionen, da die Pakete der einzelnen Klassen nacheinander rausgeschickt werden. Die Versuche 7-8 dienen nur der Kontrolle, ob die Maßnahmen für das andere Interface auch funktionieren, sie sind analog zu den Versuchen 1, 2 und 4. Versuch 10-12 sind "gemischte" Transfers, bei denen beide Interfaces

angesprochen werden. Bei Versuch 10 und 11 behindern sich die Ströme wieder gegenseitig, bei Versuch 10 wird aber noch Bandbreite von der FTP-Klasse geborgt. In Versuch 12 ist zu sehen, dass die SCP-Transfers ohne QoS immer mal wieder stehenbleiben, während mit QoS in etwa dieselben Bandbreiten garantiert werden können wie in Versuch 6. Lediglich das FTP bricht etwas ein, wahrscheinlich weil es nicht auf das selbe Interface wie die SCP's geschickt wird (der Reihe nach!), sondern über das andere, und wegen der auftretenden Kollisionen keine höhere Rate erreicht.

4.3.5. Versuch: Vermeidung von Slow Starts mit RED

Szenario

Die Rechner *aramis* und *porthos* senden gleichzeitig einen TCP-Datenstrom an *artagnan* (172.16.23.2). Da die Verbindung damit überlastet ist, werden ständig Kollisionen festgestellt, was dazu führt, dass die TCP-Flows in den Slow-Start-Modus geraten. Als Ausweg soll hier die QDisc RED dienen.

Frage 4.3.5.1:
Was erwarten Sie von dieser Maßnahme?

Die effektiv erreichte Bandbreite könnte etwas steigen und wird evtl. geglättet (beide Rechner zusammen betrachtet), da jetzt meist nicht beide gleichzeitig in den Slow-Start-Modus gehen.

Aufgabe

Führen Sie obige QoS-Maßnahme durch! Sinnvolle Ergebnisse lassen sich dabei aber nur in einem belasteten Netz erzielen. Initiieren Sie dazu bitte mittels `ping -f 172.16.23.2` ein Flood-Ping von *athos* aus.

Frage 4.3.5.2:
An welchen Stellen könnten Sie RED einsetzen, d.h. in welchen Rechnern bzw. mit oder ohne CBQ? Wie beurteilen Sie die einzelnen Varianten?

Man kann RED in den Endknoten oder im Router direkt an das Interface binden, aber auch im Router mit CBQ getrennt nach Quelladressen einsetzen. Die letzte Variante ist wahrscheinlich mit etwas mehr Overhead verbunden, die ersten beiden Möglichkeiten sind also vorzuziehen. Falls sich nicht die Endnutzer darum kümmern sollen, scheint die zweite Variante die optimale zu sein. Im Versuch wird die Variante, die RED in den Endknoten einsetzt, sowieso nicht die erwarteten Resultate erzielen, da durch das Flood-Ping nur das Netz zwischen *athos* und *artagnan* belastet wird.

Wählen Sie eine oder mehrere Varianten aus und messen Sie mit FTP vor und nach dem Einsatz von RED die erreichte Bandbreite, indem Sie von beiden Rechnern gleichzeitig ein File zu *artagnan* kopieren. Ein guter Wert für die Drop-Wahrscheinlichkeit ist z.B. 0.2. Falls Sie genug Zeit finden sollten, können Sie aber auch noch andere testen. Vergleichen Sie die praktischen Ergebnisse mit Ihren Erwartungen und interpretieren Sie sie!

Hinweis!
Machen Sie möglichst mehrere Messungen, um auszuschließen, dass es sich nur um normale Schwankungen handelt, und bilden Sie jeweils den Mittelwert. Außerdem sollten die Files gleich groß sein, damit die Transfers gleich lang dauern!

4.3.6. Bemerkungen zum Versuch 3

Die Effekte beim Einsatz von RED im dritten Versuch sind nicht sehr groß. Es sollte eine leichte Erhöhung der erreichten Bandbreite festgestellt werden, da die TCP-Datenströme bei einer Überlastsituation nicht gleichzeitig in den Slow-Start-Modus gehen, sondern schon vorher. Da das aber selten bei beiden zur selben Zeit passiert, ist die Leitung besser ausgelastet. Um auszuschließen, dass es sich bei den Messwerten nur um normale Schwankungen handelt, sollten mehrere Messungen gemacht und der Mittelwert gebildet werden. Ansonsten ist der Versuch wieder recht einfach.

Musterlösung

```
# Direkt an ausgehendes Interface binden (Router oder Endknoten)
tcc red eth0 1: root 100MBit 0.2
```

```
# Variante mit CBQ
tcc cbq    eth0 1: root 100MBit
tcc class  eth0 1:1 1:0 100MBit 100MBit 10MBit 8
tcc class  eth0 1:2 1:1 100MBit  50MBit  5MBit 5  # Rechner 1
tcc class  eth0 1:3 1:1 100MBit  50MBit  5MBit 5  # Rechner 2
tcc red    eth0 2:  1:2 100MBit 0.2
tcc red    eth0 3:  1:3 100MBit 0.2
tcc filter eth0 1:2 1:0 match ip src 134.109.192.63/32
tcc filter eth0 1:3 1:0 match ip src 134.109.192.193/32
```

Variante	Dropwkt.	Rate in MB/s		
		aramis	porthos	Gesamt
ohne QoS	-	4.45	4.69	9.14
in den Endknoten	0.20	4.51	4.58	9.09
	0.10	4.50	4.61	9.11
	0.05	4.63	4.52	9.15
im Router direkt am Interface	0.20	4.65	4.80	9.45
	0.10	4.77	4.66	9.43
	0.05	4.71	4.65	9.36
im Router mit CBQ	0.20	4.79	4.88	9.67
	0.10	4.90	4.71	9.61
	0.05	4.83	4.70	9.53

Die erste Variante bringt wie erwartet keine Resultate hervor, da sich das Flood-Ping nicht auf das Netz, in dem sich die Endknoten befinden, auswirkt. Die anderen Varianten bringen beide einen Gewinn, entgegen den Vermutungen ist aber Variante 3 (mit CBQ) vorzuziehen, die erstaunlicherweise effizienter als Variante 2 ist - trotz des höheren Aufwandes, wahrscheinlich, da die Pakete differenzierter weggeworfen werden (nach Datenströmen unterteilt). Da es sich bei der verwendeten Netztechnologie um ein Broadcast-Medium handelt, wäre es auch denkbar, zwei Datenströme zwischen jeweils zwei verschiedenen Rechnern laufen zu lassen, damit nicht nur ein Rechner alles empfangen muss. Allerdings ließen sich dabei keine anderen Ergebnisse feststellen.

4.3.7. Versuch: Kanalbündelung mit TEQL

Szenario

Der Rechner *artagnan* besitzt 3 Netzinterfaces, von denen 2 (*eth0* und *eth1*) gemeinsam genutzt werden sollen, um eine höhere Bandbreite zu erreichen.

Frage 4.3.7.1:
Welche anderen Möglichkeiten fallen Ihnen ein, um beide Leitungen gleichzeitig nutzen zu können (weniger in Endknoten, sondern eher in großen Routern eingesetzt)?

Dynamisches Routing, RIP, OSPF, ...

Aufgabe

Benutzen Sie die Queuing Discipline TEQL, um den Traffic auf die beiden Interfaces zu verteilen! Nach dem Hinzufügen an beide Interfaces geben Sie dem Device *teql0* bitte die selbe IP wie einem der beteiligten Interfaces, aber mit einer Netzmaske von 255.255.255.255 und einer Broadcast-Adresse, die der IP entspricht. Löschen Sie die Routen auf die beteiligten Interfaces und leiten Sie allen Traffic über *teql0*!

Frage 4.3.7.2:
Warum wird man bei diesem Szenario TEQL eher auf dem Server als im Client einsetzen?

Weil beim Download von Webseiten wesentlich mehr Bandbreite benötigt wird als für den HTTP-Request. Bei einem FTP-Upload z.B. wäre das allerdings genau umgekehrt.

Rufen Sie von *athos* aus eine Webseite von *artagnan* ab und lauschen Sie auf *athos* mit `tcpdump -p -l -i INTERFACE | grep http` oder auch mittels `tcpdump -p -i INTERFACE port http` an beiden Interfaces, um zu sehen, welchen Weg die Webseite nimmt. Da TEQL manchmal erst in einer Überlastsituation zu greifen scheint, funktioniert das ab und zu nicht auf Anhieb. Setzen Sie dann bitte von *athos* aus das Netz mit `ping -f -s 16000 artagnan` unter Last und probieren Sie es noch einmal. Benutzen Sie auf *artagnan* `watch -n 2 ifconfig`, um festzustellen, wieviele Pakete auf welchen Interfaces empfangen und gesendet werden. Wenn Sie dann noch Zeit haben, können Sie auch noch eine NetPIPE-Messreihe starten (ohne Flood-Ping!), um zu sehen, ob sich die effektive Bandbreite erhöht hat. Optimal wäre es dann natürlich, wenn Sie auf *athos* ebenfalls TEQL einsetzen würden - mit einer bis auf die IP identischen Konfiguration. Was haben Sie für Resultate erhalten?

4.3.8. Bemerkungen zum Versuch 4

Die Queuing Discipline TEQL kann eingesetzt werden, um Datenströme über mehrere Netzinterfaces nach außen zu leiten. Damit ließen sich z.B. mehrere Einwahlleitungen simulieren. Sinnvollerweise sollte man TEQL jedoch auf einem Server einsetzen, da zumindest bei Webservern wahrscheinlich mehr Downloads als Uploads stattfinden und sich deshalb ein Einsatz dort mehr lohnt. Mit den gegebenen Hinweisen lassen sich hier sehr schnell Resultate erzielen, weswegen man durchaus noch die NetPIPE-Messreihen in Angriff nehmen sollte.

Dieser Versuch hat bei der Ausarbeitung am meisten Zeit gekostet, weil die Dokumentation zu TEQL noch spärlicher ausgefallen ist als zu den anderen Queuing Disciplines. Außerdem ließen sich die erwarteten Ergebnisse nicht immer beobachten, sondern manchmal nur, wenn das Netz unter Last gesetzt wurde. Aufgetreten ist das z.B. beim Abruf von Webseiten über das TEQL-Interface, aber auch beim Einsatz von NetPIPE - meist nicht reproduzierbar. Bei NetPIPE trat zusätzlich noch der Effekt auf, dass die beteiligten physischen Interfaces bei kleinen Blockgrößen abwechselnd genutzt wurden.

Musterlösung

```
tcc teql eth0
tcc teql eth1
ifconfig teql0 134.109.192.178 netmask 255.255.255.255 broadcast
134.109.192.178 up
route del -net 172.16.23.0 netmask 255.255.255.0 eth0
[ route del -net 134.109.192.0 netmask 255.255.255.0 eth1 ]
route del default
route add default teql0
```

Wie erwartet, verlassen die Pakete *artagnan* auf beiden Interfaces und erreichen *athos* ebenfalls auf beiden, allerdings manchmal erst nach einem Flood-Ping. Bei NetPIPE erreicht man bis zu 105MBit/s, wenn nur auf einem Rechner TEQL eingesetzt wird, wenn es auf beiden benutzt wird, dann sogar 112-150MBit/s (Maximum bei 32KByte Blockgröße).

5. Abschließende Bemerkungen

QoS and/or fair queueing bietet die Möglichkeit, ohne zusätzliche Hard- oder Software von einem Linux-Rechner aus Quality of Service zu realisieren. Zu diesem Zweck gibt es eine Fülle von Algorithmen, die auch unter Volllast und im Langzeit-Versuch noch sehr stabil laufen. Allerdings sollte die Anzahl der Serviceklassen gering gehalten werden, weswegen einige Queuing Disciplines unter Umständen nicht für größere Netze geeignet sind. Um die QoS-Möglichkeiten auf breiter Basis einsetzen zu können, fehlt auf alle Fälle noch eine bessere Dokumentation, was sich sicherlich mit dem neuen 2.4-er Linux-Kern bessern wird.

In den Praktikums-Versuchen habe ich die meiner Meinung nach wichtigsten Algorithmen herangezogen und in Szenarien angewendet, die prinzipiell so oder so ähnlich auch in der Realität vorkommen können. Weitere Versuche können aber natürlich jederzeit in das Praktikum eingefügt werden.

6. Quellen

Paul Ferguson, Geoff Huston:

"Quality of Service - Delivering QoS on the Internet and in Corporate Networks"
http://www.wiley.com/compbooks/catalog/24358-2.htm

Wiley Computer Publishing, 1998

Mario Lorenz: "Geordnete Wege - Traffic Control mit Linux", iX 4/2000, Verlag Heinz Heise GmbH &Co KG

Saravanan Radhakrishan: "Internet Protocol - Quality of Service Page"
http://qos.ittc.ukans.edu/

Saravanan Radhakrishan: "IP Quality of Service: An Overview"
http://www.ittc.ukans.edu/~rsarav/ipqos/ip_qos.htm

Saravanan Radhakrishan: "Linux - Advanced Networking Overview" (30.9.1999)
http://qos.ittc.ukans.edu/howto/

Bert Hubert, Gregory Maxwell, Remco van Mook, Martijn van Oosterhout, Paul B Schroeder: "Linux 2.4 Advanced Routing HOWTO" (20.5.2000)
http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html

Anand Iyer: "A Study of IP QoS"
http://www.ittc.ukans.edu/~iyer/Mainpage.html

Saravanan Radhakrishan: "Linux QoS Support" (18.03.1999)
http://www.ittc.ukans.edu/~rsarav/projects/networking/ipqos/diffoverview/

Saravanan Radhakrishan: "IP QoS Efforts" (14.05.1999)
http://qos.ittc.ukans.edu/slides/

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

Alexey Kuznetsov: README zu IPRoute2 und TC
file:///usr/doc/packages/iproute/README.iproute2+tc

Alan ?: "Traffic Shaper For Linux"
file:///usr/src/linux/Documentation/networking/shaper.txt

Linux-Kernelquellen
file:///usr/src/linux/net/sched/

A. Listings

A.1. Von Seite 18 (3.4): Vollständiger Quellcode zu tcc

```
#!/bin/sh

TC="/usr/sbin/tc"
TCC=‘basename $0‘
MODPROBE="/sbin/modprobe"

CMD=$1
shift

if [ "$3" ]; then
  PARENT=$3
  test "$PARENT" = root || PARENT="parent $3"
fi

case $CMD in

  cbq)
    if [ -z $4 ]; then
      echo "Usage: ${TCC} cbq DEVICE HANDLE { root | PARENTID } BANDWIDTH"
    else
      echo "Adding qdisc $CMD on device $1 with handle $2 for $PARENT..."
      ${MODPROBE} sch_$CMD || echo "No such module"
      ${TC} qdisc add dev $1 handle $2 $PARENT $CMD \
      bandwidth $4 avpkt 1000 mpu 64 cell 8
    fi
    ;;

  red)
    if [ -z $5 ]; then
      echo "Usage: ${TCC} red DEVICE HANDLE { root | PARENTID } BANDWIDTH PROBABILITY"
    else
      echo "Adding qdisc $CMD on device $1 with handle $2 for $PARENT..."
```

```
    ${MODPROBE} sch_$CMD || echo "No such module"
    ${TC} qdisc add dev $1 handle $2 $PARENT $CMD \
    bandwidth $4 limit 60KB min 15KB max 45KB avpkt 1000 \
    burst 20 probability $5
  fi
  ;;

sfq)
  if [ -z $3 ]; then
    echo "Usage: ${TCC} sfq DEVICE HANDLE { root | PARENTID }"
  else
    echo "Adding qdisc $CMD on device $1 with handle $2 for $PARENT..."
    ${MODPROBE} sch_$CMD || echo "No such module"
    ${TC} qdisc add dev $1 handle $2 $PARENT $CMD \
    perturb 15 quantum 1514
  fi
  ;;

tbf)
  if [ -z $4 ]; then
    echo "Usage: ${TCC} tbf DEVICE HANDLE { root | PARENTID } RATE"
  else
    echo "Adding qdisc $CMD on device $1 with handle $2 for $PARENT..."
    ${MODPROBE} sch_$CMD || echo "No such module"
    ${TC} qdisc add dev $1 handle $2 $PARENT $CMD \
    buffer 10Kb/8 limit 15Kb rate $4
  fi
  ;;

teql)
  if [ -z $1 ]; then
    echo "Usage: ${TCC} teql DEVICE"
  else
    echo "Adding qdisc $CMD on device $1..."
    ${MODPROBE} sch_$CMD || echo "No such module"
    ${TC} qdisc add dev $1 root teql0
  fi
  ;;

class)
  if [ -z $7 ]; then
    echo "Usage: ${TCC} class DEVICE CLASSID PARENTID \\"
    echo "       BANDWIDTH RATE WEIGHT PRIO [ bounded ] [ isolated ]"
  else
    echo "Adding class $2 on device $1 for parent $3..."
    ${TC} $CMD add dev $1 classid $2 parent $3 cbq \
```

```
    bandwidth $4 avpkt 1000 mpu 64 cell 8 allot 1514 \
    rate $5 weight $6 maxburst 20 prio $7 $8 $9
  fi
  ;;

filter)
  if [ -z $4 ]; then
    echo "Usage: ${TCC} filter DEVICE FLOWID PARENTID match SELECTOR"
  else
    echo "Adding filter on device $1 for parent $3 and flowid $2..."
    ${MODPROBE} cls_u32 || echo "No such module"
    FDEV=$1; FFLOW=$2; FPARENT=$3
    shift; shift; shift
    ${TC} $CMD add dev $FDEV parent $FPARENT \
    protocol ip prio 100 u32 $* flowid $FFLOW
  fi
  ;;

remove|rm)
  if [ -z $1 ]; then
    echo "Usage: ${TCC} remove DEVICE"
  else
    echo "Removing all qdiscs, classes, and filters from device $1..."
    ${TC} qdisc del dev $1 root 2>/dev/null
  fi
  ;;

list|ls)
  if [ -z $1 ]; then
    echo "Usage: ${TCC} list DEVICE [ -s ]"
  else
    echo "Queuing disciplines:"
    ${TC} $2 qdisc ls dev $1
    echo ""
    echo "Assigned classes:"
    ${TC} $2 class ls dev $1
    echo ""
    echo "Classifying filters:"
    ${TC} filter ls dev $1
  fi
  ;;

*)
  echo "Usage: ${TCC} { cbq | red | sfq | tbf | teql | class | filter } OPTIONS"
  echo "       ${TCC} { remove | list } DEVICE"
  ;;
```

```
esac
```

A.2. Von Seite 18 (3.4): Vollständiger Quellcode zu netpipe

```
#!/bin/sh

NETPIPE="/usr/local/bin/NPtcp"
MESSDIR="/root/prak_messungen/"
BUFFER="65536"
LOWER="1"
UPPER="2097152"

case $1 in

  send)
    ${NETPIPE} -t -h $2 -p $3 -l ${LOWER} -u ${UPPER} -b ${BUFFER} \
               -o ${MESSDIR}$4 -P
    ;;

  recv)
    ${NETPIPE} -r -p $2 -l ${LOWER} -u ${UPPER} -b ${BUFFER}
    ;;

  *)
    echo "Usage: netpipe send HOST PORT MESSFILE"
    echo "       netpipe recv PORT"
    ;;

esac
```

A.3. Von Seite 19 (3.4): Vollständiger Quellcode zu routes

```
#!/bin/sh

case $1 in

  set)
    echo "Setting routes..."

    for i in artagnan; do
      ssh root@$i /sbin/route del default
      ssh root@$i /sbin/route del -net 134.109.192.0 netmask 255.255.255.0 eth1
      ssh root@$i /sbin/route add default gw 172.16.23.1 eth0
    done

    for i in athos; do
      ssh root@$i /sbin/route del default
      ssh root@$i /sbin/route add default eth1
    done

    for i in aramis porthos; do
      ssh root@$i /sbin/route del default
      ssh root@$i /sbin/route add default gw 134.109.192.190 eth0
    done
    ;;

  reset)
    echo "Resetting routes..."

    for i in artagnan; do
      ssh root@$i /sbin/route del default
      ssh root@$i /sbin/route add -net 134.109.192.0 netmask 255.255.255.0 eth1
      ssh root@$i /sbin/route add default gw 192.168.42.254 eth2
    done

    for i in athos; do
      ssh root@$i /sbin/route del default
      ssh root@$i /sbin/route add default gw 192.168.47.254 eth2
    done

    for i in aramis porthos; do
      ssh root@$i /sbin/route del default
      ssh root@$i /sbin/route add default gw 192.168.26.254 eth1
    done
```

```
    ;;

  list)
    for i in athos artagnan aramis porthos; do
      echo -n "$i: "
      ssh root@$i /sbin/route -n
      echo ""
    done
    ;;

  *)
    echo "Usage: routes { set | reset | list }"
    ;;

esac
```

A.4. Von Seite 19 (3.4): Vollständiger Quellcode zu prak

```
#!/bin/sh

case $1 in

  sync)
    echo "Syncing files..."

    for i in aramis artagnan porthos; do

      echo "------------------------------------------------------------"
      echo "Copying files to $i..."
      echo "------------------------------------------------------------"

      scp -r /root/praktikum /root/.bashrc \
          root@$i:/root

      scp /boot/System.map /boot/System.map.old \
          /boot/vmlinuz /boot/vmlinuz.old \
          root@$i:/boot

      scp /etc/lilo.conf /etc/ftpusers /etc/hosts \
          /etc/inetd.conf /etc/resolv.conf /etc/profile.local \
          root@$i:/etc

      scp -r /lib/modules \
          root@$i:/lib

      echo "------------------------------------------------------------"
      echo "Calling LILO..."
      echo "------------------------------------------------------------"

      ssh root@$i /sbin/lilo

    done
    ;;

  pack)
    echo "Packing /root/praktikum..."
    tar cvzf /root/praktikum.tar.gz /root/praktikum
    ;;

  reboot)
```

```
    for i in aramis artagnan porthos athos; do
      ssh root@$i /sbin/shutdown -r now
    done
    ;;

  halt)
    for i in aramis artagnan porthos athos; do
      ssh root@$i /sbin/shutdown -h now
    done
    ;;

  *)
    echo "Usage: prak { sync | pack | reboot | halt }"
    ;;

esac
```