

ITM-Praktikum

Versuch 6: Quality of Service, Traffic Shaping

Andreas Klingler, Hannes Stahl, Simon Lüke

6. Juli 2009

Inhaltsverzeichnis

1	Vorbereitende Fragen	2
1.1	Quality of Service (QoS)	2
1.2	Aufteilung in Serviceklassen (CoS)	2
1.3	Anforderungen an QoS	2
1.4	grundlegende Verfahren	2
1.4.1	Admission Control	2
1.4.2	Traffic Shaping	2
1.4.3	Preferential Queuing	2
1.4.4	Selective Forwarding	2
1.4.5	TCP-Mechanismen	3
1.5	Classless Queuing Disciplines	3
1.5.1	FiFo Queue	3
1.5.2	pFiFo Queue	3
1.5.3	SQF (Stochastic Fair Queueing)	3
1.6	Classful Queuing Disciplines	3
1.6.1	PRIO qdisc	3
1.6.2	CBQ	3
1.6.3	HTB	4
1.7	Unterschiede zwischen DiffServ und IntServ	4
1.7.1	DiffServ (Differentated Services)	4
1.7.2	IntServ (Integrated Services)	4
2	Versuchsdurchführung	4
2.1	TOS-Feld / DS-Feld	4
2.1.1	HTTP Session	4
2.1.2	SSH Sitzung	4
2.1.3	FTP Transfer	4
2.2	QoS mit CBQ	4
2.2.1	ohne QoS, ohne Netzüberlastung	6
2.2.2	ohne QoS, mit Netzüberlastung	6
2.2.3	mit Qos, mit Netzüberlastung	6
2.2.4	mit QoS, ohne Netzüberlastung	6
2.3	QoS mit HTB	6
2.3.1	QoS ohne Überlast	7
2.3.2	QoS mit Überlast	7
2.4	Bandbreitenmessung mit Iperf	8

1 Vorbereitende Fragen

1.1 Quality of Service (QoS)

QoS beschreibt die Dienstgüte, also inwiefern die Qualität einer Verbindung mit den Anforderungen übereinstimmt. Der QoS wird in z. B. IP Netzen durch folgende Parameter erfasst:

- Latenzzeit
- Jitter
- Droprate
- Datenrate

1.2 Aufteilung in Serviceklassen (CoS)

Während QoS versucht, für die Verbindung spezielle Randbedingungen zu erfüllen, teilt CoS die Pakete in drei Klassen ein wodurch eine gezielte Priorisierung möglich ist. Die drei Klassen von CoS sind folgendermaßen eingeteilt:

- Klasse 1: Sprache
- Klasse 2: geschäftskritischer Datentransfer wie SAP oder Videokonferenzen
- Klasse 3: geschäftsunkritische Anwendungen wie E-Mail und Webbrowser

1.3 Anforderungen an QoS

Mittels QoS sollen die Anforderungen an die Verbindung messbar sein, so lassen sich Qualitätsmerkmale wie Jitter oder Bandbreite vergleichen.

1.4 grundlegende Verfahren

1.4.1 Admission Control

Ein Dienst beschreibt in einem „Traffic Contract“ seine QoS-Anforderungen. die „Admission Control“ kann diesen bei ausreichenden Ressourcen zustimmen.

1.4.2 Traffic Shaping

Wie der Name schon sagt, soll der Verkehr „geformt“ werden, so dass bestimmte Anforderungen an das Netz wie z. B. niedrige Latenz, hohe Performance, hohe Bandbreite, etc. erfüllt werden.

1.4.3 Preferential Queuing

Bei „Preferential Queuing“ werden nicht alle Datenpakete gleichbedeutend behandelt, sondern eine Priorisierung eingeführt. Diese soll möglichst so erfolgen, dass Dienste die eine hohe QoS erfordern bevorzugt behandelt werden.

1.4.4 Selective Forwarding

Mit „Selective Forwarding“ kann Datenverkehr selektiv weitergeleitet oder gefiltert werden. Forwarding findet vor dem Queuing statt.

1.4.5 TCP-Mechanismen

- RED (Random Early Detection): Es reguliert den Drop von eigenhändigen Paketen anhand des Füllstandes des Puffers. Ist der Puffer leer, werden keine Pakete gedropt. Füllt er sich langsam, so werden immer mehr Pakete verworfen, bei vollem Puffer ist die Wahrscheinlichkeit des Drops 1.
- WRED (Weighted Random Early Detection): Hier ist die Droprate von der Gewichtung/Priorität abhängig
- GRED (Generalized RED): Hier ist die Dropwahrscheinlichkeit von der Queue abhängig.

1.5 Classless Queuing Disciplines

Bei klassenlosem Queueing werden Datenpakete lediglich verzögert, neu angeordnet oder gedropt. Dies kann auf unterschiedlichen Arten in unterschiedlichen Queues passieren:

1.5.1 FiFo Queue

Die FiFo (First in First Out) Queue verarbeitet die Daten nacheinander, so wie sie zeitlich eingegangen sind. Sie kann also die Daten vor dem Senden nicht neu anordnen.

1.5.2 pFiFo Queue

Hier gibt es mehrere unterschiedlich Priorisierte Queues, in welche die Daten einsortiert und versendet werden. Diese Queue gilt als klassenlos, weil sie aus Usersicht nicht beeinflussbar ist und weitere qdiscs eingebaut werden können.

1.5.3 SQF (Stochastic Fair Queueing)

Bei SQF bekommt jede Anwendung auf TCP Ebene eine eigenen Queue. Durch „Zufall“ werden den unterschiedlichen Queues Sendekapazitäten gewährt (z. B. 1514 Byte lang).

1.6 Classful Queuing Disciplines

„Classful Queuing Disciplines“ erlaubt das hinzufügen unterschiedlicher qdiscs oder weiterer Klassen, die wiederum qdiscs oder Klassen enthalten können. So ist die Queue aus Usersicht beeinflussbar, man kann sich selber seine Queue zusammenstellen. Tritt ein Paket in die Ingress Queue ein, muss es also zuerst „klassifiziert“ werden um es der Richtigen Queue zuzuordnen.

1.6.1 PRIO qdisc

Die PRIO Queue arbeitet ähnlich wie die pFiFo Queue, allerdings sind die einzelnen Prioritätsbänder selbst Klassen. So kann man z. B. Anzahl der Bänder und die Queueart in jeden Band bestimmen.

1.6.2 CBQ

CBQ ist eine extrem komplexe Implementierung einer Queueklasse, welche mit sehr vielen Parametern gesteuert werden kann. Ihr können nun wiederum klassenlose Queues angefügt werden.

1.6.3 HTB

HTB („Hierarchical Token Bucket“) ist eine etwas einfacher gehaltene Queue, die CBQ stark ähnelt, aber mit weniger Parameter auskommt.

1.7 Unterschiede zwischen DiffServ und IntServ

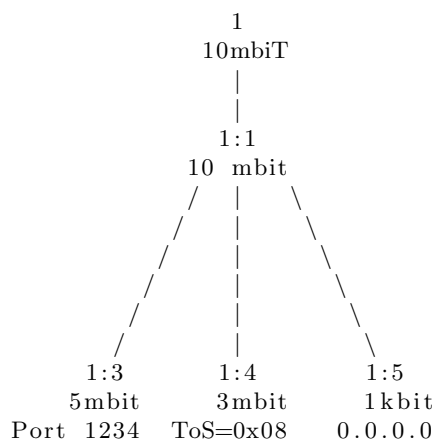
1.7.1 DiffServ (Differentated Services)

Bei diesem Verfahren werden die Pakete in verschiedene Klassen eingeteilt und vom Sender im IP Frame eingetragen (ToS (Type of Service) Feld, 6 Bit). Außerdem kann in diesem Feld noch die Drop-Precendee angegeben werden. Die Auswertung und Behandlung von diesem Feld, bleibt dabei den Routern selbst überlassen.

1.7.2 IntServ (Integrated Services)

Bei IntServ reserviert sich der Sender für jede Verbindung die benötigte Kapazität bei den Routern. Diese können diese zuteilen oder verweigern. Wird die Bandbreite zugesichert, so muss diese in jedem Fall über die komplette Sitzungslänge zur Verfügung stehen. Bei diesem Verfahren muss IntServ auf jedem Router implementiert sein, sonst können andere Router nicht nach ihren gewünschten Kapazitäten nachfragen und reservieren.

1.8 Baumstruktur der QoS Konfiguration



1.9 QoS an der Uni Ulm

Die Uni Ulm wird vermutlich kein QoS machen müssen, da die Standleitungen ausreichend groß dimensioniert sind und diese somit (noch) nicht an ihre Vollausslastung kommen.

2 Versuchsdurchführung

2.1 TOS-Feld / DS-Feld

2.1.1 HTTP Session

Bei der Browsersitzung war im DSCP Feld durchweg 0x00 eingetragen. Die Daten benötigen also keinen speziellen QoS, einige ms Delay fallen beim browsen nicht auf, genauso wie geringe

Priorität beim Datendurchsatz.

2.1.2 SSH Sitzung

Bei der SSH Sitzung ist beim Verbindungsaufbau und beim Key Exchange kein Bit im DSCP Feld gesetzt. Erst wenn die Verbindung steht wird 0x04 (000100 = Min. Delay) gesetzt. Dies ist auch logisch, da der User bei einer Terminalsitzung geringe Verzögerungen bemerken würde weil die Zeicheneingabe hakelig wird.

2.1.3 FTP Transfer

Bei Steuerbefehlen wie „LIST“ wurde im DSCP Feld die Bitkombination 000100 (Minimales Delay) verwendet, während beim eigentlichen Datenversand 000010 (Max. Throughput) eingestellt war. Auch das ist leicht zu erklären: Beim Browsen durch die Ordnerstruktur soll flott der neue Ordner angezeigt werden, während beim reinen kopieren der Daten ein möglichst hoher Durchsatz gefordert ist.

2.2 QoS mit CBQ

Nun spielen wir einige Fälle mit der CBQ Queue durch. Unser Skript dafür sah folgendermaßen aus:

```
#!/bin/bash
#
# This script is for changing the traffic control settings.
#

DEVICE="eth2"

case "$1" in
    start)
        tc qdisc add dev $DEVICE root handle 1: cbq bandwidth 10mbit avpkt 1000
            cell 8

        # prio : In the round-robin process, classes with the lowest priority
            field are tried for packets first.
        tc class add dev $DEVICE parent 1: classid 1:1 cbq bandwidth 10mbit
            rate 8mbit weight 0.8mbit prio 8 allot 1514 cell 8 maxburst 20
            avpkt 1000 bounded

        tc class add dev $DEVICE parent 1:1 classid 1:3 cbq bandwidth 10mbit
            rate 5mbit weight 0.5mbit prio 4 allot 1514 cell 8 maxburst 20
            avpkt 1000

        tc class add dev $DEVICE parent 1:1 classid 1:4 cbq bandwidth 10mbit
            rate 3mbit weight 0.3mbit prio 2 allot 1514 cell 8 maxburst 20
            avpkt 1000

        tc class add dev $DEVICE parent 1:1 classid 1:5 cbq bandwidth 10mbit
            rate 1kbit weight 0.1kbit prio 8 allot 1514 cell 8 maxburst 20
            avpkt 1000

        tc qdisc add dev $DEVICE parent 1:3 handle 30: sfq perturb 10
        tc qdisc add dev $DEVICE parent 1:4 handle 40: sfq perturb 10
```

```

tc qdisc add dev $DEVICE parent 1:5 handle 50: sfq perturb 10

# prio : The priority of this classifier. Lower numbers get tested
# first.
tc filter add dev $DEVICE parent 1:0 protocol ip prio 2 u32 match ip
dport 1234 0xffff flowid 1:3
tc filter add dev $DEVICE parent 1:0 protocol ip prio 1 u32 match ip
tos 0x08 0xff flowid 1:4
tc filter add dev $DEVICE parent 1:0 protocol ip prio 3 u32 match ip
src 0.0.0.0/0 flowid 1:5
;;
stop)
tc qdisc del dev $DEVICE root
;;
restart)
$0 stop
$0 start
;;
info)
tc -s class show dev $DEVICE
;;
*)
echo "Usage: $0 {start|stop|restart|info}"
exit 1
;;
esac

```

2.2.1 ohne QoS, ohne Netzüberlastung

Im ersten Fall wird kein QoS verwendet. Ein FTP Transfer erreicht ca. 870 kB/s wenn nicht gestreamt wird. Mit Streaming bricht der FTP Transfer etwas ein, das Bild ist gut erkennbar.

2.2.2 ohne QoS, mit Netzüberlastung

Von Fall 1 ausgehend simulieren wir nun eine Überlastsituation mit einem Flooding. Bei einem Flood werden hin- und wieder große Pixel am Video sichtbar. Bei zwei Flooding ist schon gar kein Videobild mehr zu erkennen.

2.2.3 mit QoS, mit Netzüberlastung

Bei leichter Netzüberlastung mit FTP, Stream und einem Flooding bleibt das Video weiterhin gut, der FTP Transfer bricht auf 250kB/s ein. Bei zwei Flooding bricht der FTP Transfer ab, das Video wird zwar pixelig aber erkennbar. Somit sind erste Resultate des QoS „sichtbar“, wobei der Verbindungsabbruch des FTP Transfers nicht gerade schön ist.

2.2.4 mit QoS, ohne Netzüberlastung

Ohne Netzüberlastung und QoS sind leichte Performanceeinbußen gegenüber Fall 1 zu beobachten:

- Der reine FTP Download fällt von 870kB/s auf 800kB/s bei eingeschaltetem QoS

- Bei kombiniertem FTP und Streamingverkehr ist das Video zwar gut, aber der FTP Transfer verzeichnete leichte Einbußen gegenüber Fall 1.

2.3 QoS mit HTB

Das Skript mit der HTB Queue braucht etwas weniger Angaben, hat aber ähnliche Syntax:

```
#!/bin/bash
#
# This script is for changing the traffic control settings.
#

DEVICE="eth2"

case "$1" in
    start)
        tc qdisc add dev $DEVICE root handle 1: htb default 50

        # prio : In the round-robin process, classes with the lowest priority
        # field are tried for packets first.
        tc class add dev $DEVICE parent 1: classid 1:1 htb rate 8mbit ceil
        10mbit burst 30k

        tc class add dev $DEVICE parent 1:1 classid 1:3 htb rate 5mbit ceil
        8mbit burst 30k
        tc class add dev $DEVICE parent 1:1 classid 1:4 htb rate 3mbit ceil
        8mbit burst 30k
        tc class add dev $DEVICE parent 1:1 classid 1:5 htb rate 1kbit ceil
        8mbit burst 30k

        tc qdisc add dev $DEVICE parent 1:3 handle 30: sfq perturb 10
        tc qdisc add dev $DEVICE parent 1:4 handle 40: sfq perturb 10
        tc qdisc add dev $DEVICE parent 1:5 handle 50: sfq perturb 10

        # prio : The priority of this classifier. Lower numbers get tested
        # first.
        tc filter add dev $DEVICE parent 1:0 protocol ip prio 2 u32 match ip
        dport 1234 0xffff flowid 1:3
        tc filter add dev $DEVICE parent 1:0 protocol ip prio 1 u32 match ip
        tos 0x08 0xff flowid 1:4
        tc filter add dev $DEVICE parent 1:0 protocol ip prio 3 u32 match ip
        src 0.0.0.0/0 flowid 1:5
        ;;
    stop)
        tc qdisc del dev $DEVICE root
        ;;
    restart)
        $0 stop
        $0 start
        ;;
    info)
        tc -s class show dev $DEVICE
        ;;
    *)
        ;;
esac
```



```

*)
  echo "Usage: $0 {start|stop|restart|info}"
  exit 1
;;
esac

```

2.3.1 QoS ohne Überlast

Ein einzelner FTP Download erreichte nun wieder die Performance wie zuvor ohne QoS (870kB/s), ebenso war ein Videostream kein Problem.

Ein zusätzlicher einzelner Flooding ließ den FTP Transfer auf 300kB/s einbrechen. Das ist immerhin fast doppelt so viel Durchsatz wie es CBQ zuvor in dieser Situation zuließ.

2.3.2 QoS mit Überlast

Im letzten Fall wurde die Leitung mit zwei Flooding überlastet. Das Video blieb weiterhin perfekt in der Qualität und der FTP Transfer ging auf 240kB/s zurück! Ein respektables Ergebnis, wenn man bedenkt, dass CBQ zwar die Videoqualität erhalten konnte aber dafür den FTP Transfer zusammenbrechen ließ. Somit war bei unsern Tests HTB die bessere Wahl.

2.4 Bandbreitenmessung mit Iperf

Nun messen wir den Datendurchsatz über die einzelnen Queues bei unterschiedlichen Auslastungen. Zuerst schicken wir nur einen Transfer auf Port 1234:

```
iperf -c address -i 5 -t 60 -p 1234
```

dieser Transfer ergab einen Durchsatz von fast 8 Mbit. Die Queue an Port 1234 sollte 5 Mbit bekommen, darf aber bei unausgelasteter Leitung bis zu 8Mbit belegen, was nun der Fall war.

Nun senden wir an zwei Queues gleichzeitig Daten:

```
iperf -c address --tos 0x08
iperf -c address -p 1234
```

Theoretisch sollte nun der Bulktraffic mit tos=0x08 3Mbit bekommen, während der Traffic an Port 1234 laut Skript 5Mbit bekommen sollte.

Tatsächlich bekam der Bulktraffic 3,23Mbit und der andere Traffic bekam 4,33Mbit. Die Messwerte passen also grob zu unserer Konfiguration, die Abweichung könnte von den unterschiedlichen Startzeitpunkten des Traffics oder der Ungenauigkeit von Iperf und HTB stammen.

Im letzten Fall nutzen wir alle drei Queues aus:

```
iperf -c address --tos 0x08
iperf -c address -p 1234
iperf -c address -p 5002
```

Folgende Messwerte wurden von Iperf ermittelt:

- Traffic an Port 1234: 4,01 Mbit (Theorie: 5Mbit)
- Traffic an mit ToS=0x08: 3,3Mbit (Theorie: 3Mbit)
- Traffic an Port 5002: 340kBit (Theorie: 0,1Mbit)

Insgesamt also wieder leichte Abweichungen, aber im Großen und Ganzen passt die Bandbreitenaufteilung zu unserem vorgegebenem Traffic Shape.