

Layer 2 Link Emulation in virtuellen Netzen

Studienprojekt
im
Studiengang Systems Engineering
am Institut für Informatik und Wirtschaftsinformatik
der Universität Duisburg-Essen

Johannes Formann

2202696

Essen, 20.08.2010

Betreuer: Martin Becke

Zusammenfassung

Dokumentation der Konzeption und Entwicklung eines Prototypen zur Emulation von Layer 2 Verbindungen unter QoS Aspekten. Die gewonnenen Erkenntnisse werden präsentiert und mit einer bereits existierenden Implementierung verglichen.

Danksagung

Ich bedanke mich für die Unterstützung bei der Erstellung der Arbeit durch den Lehrstuhl Technik der Rechnernetze, insbesondere bei dem Betreuer der Arbeit Martin Becke.

Abbildungsverzeichnis

1	Testaufbau	5
2	Netfilter-Hooks im Linuxkernel	15
3	Referenzwerte für die Verzögerung bei verschiedenen Paketgrößen	17
4	Referenzwerte für den Paketverlust in Prozent ohne Emulationstechniken bei verschiedenen Bandbreiten	18
5	Paketverlust in Prozent bei Konfiguration von TC/NetEm auf 100 MBit/s	19
6	Paketverlust in Prozent bei Konfiguration von DummyNet auf 100 MBit/s	20
7	CPU-Nutzung in Prozent bei geladenen und auf 100 MBit/s konfigurierten DummyNet für verschiedenen Bandbreiten	21
8	Abweichung von der konfigurierten Bandbreite von 8389 KBit/s bei verschiedenen Paketgrößen.	23
9	Vergleich der Latenz bei verschiedenen Paketgrößen und Bandbreiten mit dem theoretischen Bandbreitenlatenzprodukt	24
10	Paketverluste bei verschiedenen Bandbreiten und vorgegebenen Paketverlustraten	25
11	Paketverluste bei verschiedenen Bandbreiten und vorgegebenen Paketverlustraten. DummyNet mit einem Bandbreitenlimit von 100 MBit/s.	26
12	Modellierung von Jitter mit DummyNet. Ziel 100ms Delay mit einer Normalverteilung ± 50 ms	28
13	Modellierung von Jitter mit NetEm. Ziel 100ms Delay mit einer Normalverteilung ± 50 ms	29

Tabellenverzeichnis

- 1 Für die Messungen genutzte Software 6
- 2 Paketverlustraten bei einem Stichprobenumfang von 10.000 Paketen und
verschiedenen Ziel-Paketverlustraten 25
- 3 Delay nach Herausrechnen der Verzögerung im System ohne Emulations-
software für verschiedene vorgegebene Verzögerungen bei Paketen von 170
Bytes 27

Abkürzungsverzeichnis

CRC Cyclic Redundancy Check / Zyklische Redundanzprüfung

IP Internet Protokoll

Layer 2 Schicht 2 / Verbindungsschicht (Data Link Layer) im OSI-Referenzmodell

Layer 3 Schicht 3 / Vermittlungsschicht (Network Layer) im OSI-Referenzmodell

MAC-Adresse Media-Access-Control-Adresse

NetEm Network Emulator

PF Packet Filter

QoS Quality of Service

RTT Round Trip Time

SCTP Stream Control Transmission Protocol

TC Traffic cyclic redundancy checkic Control (Linux Implementierung)

vLAN Virtual Local Area Network

WAN Wide Adrea Networks

Inhaltsverzeichnis

I	Einleitung	1
1	Vorwort	1
2	Motivation	1
II	Techniken	2
3	Bestehende Lösungen zur Emulation von Netzwerkverbindungen	2
3.1	Einfache Paketfilter	2
3.2	Traffic Control und NetEm	2
3.3	NIST Net	3
3.4	Dummysnet	3
4	Techniken zur Analyse der QoS Parameter einer Verbindung	4
4.1	ping, traceroute, mtr & co	4
4.2	ttcp / nuttcp und iperf	4
5	Testaufbau	5
5.1	Hardware und Netzwerksetup	5
5.2	Konfiguration der Netzwerkeмуляtionssoftware	6
5.2.1	TC/NetEm	6
5.2.2	Dummysnet	8
5.3	Messungen	8
5.3.1	Datenrate/Bandwidth	9
5.3.2	Verzögerung/Delay	10
5.3.3	Paketverlustrate/Loss	10
5.3.4	Laufzeitschwankung/Jitter	11
5.3.5	Ressourcenbedarf	12
III	Implementierung	13
6	Anpassungen von Dummysnet	13
6.1	Ausgangssituation	13

6.2 Anpassungen	13
IV Tests	15
7 Erfahrungen	15
7.1 Unterschiedliche Bandbreitenberechnungen	15
7.2 Einschränkungen / Fehlfunktionen	16
8 Auswertung der Messergebnisse	17
8.1 Referenzwerte	17
8.1.1 Basissystem	17
8.1.2 TC/NetEm	18
8.1.3 Dummynet	19
8.1.4 Schlussfolgerung	20
8.2 CPU Last der verschiedenen Techniken	20
8.3 Genauigkeit der Bandbreitenlimitierung	22
8.4 Genauigkeit der Paketverlustrate	24
8.5 Genauigkeit der Verzögerung	27
8.6 Modellierung von Verzögerungsschwankungen	27
V Zusammenfassung	29
9 Zusammenfassung & Ausblick	29
9.1 Zusammenfassung	29
9.2 Ausblick	30
VI Anhang	31
A Skripte für die Messungen	31
A.1 Datenrate	31
A.2 Verzögerung	36
A.3 Paketverluste	38
A.3.1 Baseline	38
A.3.2 Vorgegebene Verlustrate	39
A.4 Verzögerungsschwankungen	46

A.5 CPU-Last	48
Literatur	53

Teil I

Einleitung

1 Vorwort

In dieser Ausarbeitung wird die Konzeption und Entwicklung eines Prototypen zur Emulation von Layer 2 Verbindungen unter QoS Aspekten dokumentiert, die gewonnenen Erkenntnisse präsentiert und mit einer bereits existierenden Implementierung verglichen.

Im Folgenden wird zunächst kurz die zugrunde liegende Idee vorgestellt, die als Motivation der Arbeit zugrunde liegt. In Teil II werden die verwendeten Techniken/Software präsentiert und im Teil III werden einige Aspekte der gewählten Umsetzung erläutert. Im Teil IV werden die durchgeführten Tests vorgestellt und ausgewertet und zum Schluss wird in Teil V ein Resümee gezogen.

2 Motivation

Für die Protokollentwicklung sind möglichst realitätsnahe Testumgebungen notwendig. Simulationen sind dafür häufig zu deterministisch und teilweise auch zu abstrakt, während bei Messungen in real existierenden Netzen die Messungen nicht immer gut wiederholbar sind¹. Emulierung von Netzwerkparametern (Verzögerung, Paketverluste, Verzögerungsschwankungen und Begrenzung der Bandbreite) zwischen zwei Systemen bietet sowohl die Wiederholbarkeit der Messung, als auch die Möglichkeit die Messung mit real existierenden Systemen durchzuführen und nur die Verbindung zu emulieren.

Um auch neue Layer 3 Protokolle evaluieren zu können, ohne die genutzte Emulationssoftware jeweils an das neue Protokoll anpassen zu müssen, muss die Link-Emulation auf Layer 2 möglich sein. Für die Protokollentwicklung werden heute zunehmend virtualisierte Systeme eingesetzt. Beispiele sind das Planetlab[13] oder auch Emulab[14]. Um in virtualisierten Systemen eingesetzt werden zu können, muss eine Link-Emulation auch mit virtuellen Netzwerkinterfaces, wie sie etwa bei Xen[9] verwendet werden, umgehen können.

Die Eignung für Link-Emulation auf Layer 2 soll in dieser Arbeit für die beiden populären Werkzeuge Dummynet[28] und dem Traffic Control Framework[18] von Linux, insbeson-

¹Die Effekte, die während eines Testlaufes auftreten, entziehen sich größtenteils dem Experimentator.

dere dabei NetEm[17, 22], untersucht werden. Die Dummynetportierung für Linux[28] muss dabei für die Messungen auf Layer 2 angepasst werden. Es soll dabei gezeigt werden, dass die Eigenschaften für Link-Emulation auf Layer 2 sich nicht von den bereits bekannten Eigenschaften auf Layer 3 unterscheiden.

Teil II

Techniken

3 Bestehende Lösungen zur Emulation von Netzwerkverbindungen

Zum Zeitpunkt der Erstellung dieser Arbeit extrapolieren sich insbesondere zwei Ansätze zur Link Emulation. Es werden zudem zwei weitere Ansätze vorgestellt, die jeweils einige Aspekte der Emulation von Netzwerkverbindungen beinhalten.

3.1 Einfache Paketfilter

Paketfilter wie netfilter/iptables[4] (Layer 3 Paketfilter), ebtables[30] (Layer 2 Paketfilter für Bridging), PF[6] (Layer 3 Paketfilter) oder ipfw[2] (Layer 2 und 3 Paketfilter) beherrschen Funktionen, um Pakete nach bestimmten Regelsätzen zu verwerfen. Es lassen sich also damit primitive Bandbreitenlimitierungen realisieren. Primitiv, weil keinerlei Queuing vorhanden ist, und somit kurze Bursts² gleich zu Paketverlusten führen und, unter Umständen, mit komplexeren Regeln auch Paketverlust modellieren. Darüber hinausgehende Funktionen sind in der Regel nicht modellierbar, sodass diese Werkzeuge trotz ihrer leichten Verfügbarkeit nicht als Link-Emulator eingesetzt werden.

3.2 Traffic Control und NetEm

Das Linux Traffic Control Framework(tc)[18] bietet vielfältige Möglichkeiten feingranular den Verkehr auf einem Interface anhand von Regelsätzen auf Layer2 und 3 zu steuern[8]. Neben der Bandbreitenlimitierung und der Möglichkeit verschiedene QoS-Modelle zu

²in diesem Kontext eine im Vergleich zur idealen Gleichverteilung gehäufte Ankunft von Paketen

implementieren, steht mit NetEm[17, 22] ein Modul zur Verfügung mit dem zusätzlich Eigenschaften von Verbindungen simuliert werden können. Dies sind Verzögerungen inklusive Streuung, Paketverluste, Paketverdopplungen, Paketbeschädigungen³ und Umsortierung. Dadurch können, wie in [17] gezeigt, sehr realistisch (WAN-)Verbindungen für bestimmte Ende-zu-Ende Verbindungen nachgebildet werden.

Für eingehende Verkehrsströme ist allerdings die Nutzung eines Pseudodevices (IFB[21]) notwendig, welches die Regelsätze verkomplizieren kann. Die Implementierung des Paketverlustmodells wird aktuell überarbeitet[29], da Untersuchungen Schwächen gezeigt haben bei der Implementierung von korrelierten Paketverlusten.

3.3 NIST Net

NIST Net[12, 5] ist ein Werkzeug zur Netzwerkemulation. Es bietet als Feature Verzögerungen, Paketverluste, Paketumsortierung und Jitter. Es ist als Kernelpatch für Linux erhältlich, und bietet nur die Möglichkeit auf Layer 3 (IP) zu arbeiten. Es scheint[20] auch nicht mehr gepflegt zu werden.

3.4 Dummynet

Dummynet[28, 10] ist ein auf FreeBSD entwickelter Link Emulator. Er ist mittlerweile auch unter Linux und Mac OS X verfügbar.

Dummynet basiert auf sogenannten Pipes. Jede Pipe hat einen Puffer, in Sonderfällen auch mehrere Puffer/Queues, welche dann anhand einer Schedulingstrategie bedient werden und eine Verzögerungsstrecke mit der Verzögerungen und Verlustwahrscheinlichkeit modelliert werden. Die Verzögerungen können dabei auch komplexen Kurven folgen, um MAC-Layer Effekte zu modellieren.

Jitter und Reordering können nicht direkt modelliert werden, sondern nur indirekt als Sammlung verschiedener Pipes mit unterschiedlichen Verzögerungen, auf die der Verkehr aufgeteilt wird.

Für ipfw[2], der als Klassifizierer für die Aufteilung der Pakete auf verschiedene Pipes genutzt wird, ist mittlerweile auch eine Erweiterung[19] für FreeBSD vorhanden, welche auf Layer 2 nach MAC-Adressen filtern kann. Damit ist Dummynet für FreeBSD sowohl auf Layer 2 als auch Layer 3 einsetzbar.

³Veränderung der übertragenen Daten

Dummysnet wird auch bereits in PlanetLab als Netzwerk-Emulator eingesetzt. Die Bandbreitenbegrenzungen werden allerdings aus organisatorischen Gründen mit dem Linux Traffic Control Framework vorgenommen[11].

4 Techniken zur Analyse der QoS Parameter einer Verbindung

Für die Bestimmung der QoS Parameter einer Verbindung gibt es zum Zeitpunkt der Erstellung dieser Arbeit eine Reihe von Werkzeugen. Im Folgenden werden die in dieser Arbeit verwendeten Werkzeuge näher vorgestellt.

4.1 ping, traceroute, mtr & co

Die Paketverluste einer Verbindung sowie die Roundtriptime und damit indirekt die Verzögerung lassen sich mit den in den meisten Systemen vorhandenen Tools wie ping und traceroute analysieren. Traceroute oder das erweiterte mtr[3] bieten zusätzlich eine Analyse für jeden feststellbaren⁴ Hop.

Um die Verzögerung zu messen sind diese Werkzeuge nur dann gut geeignet, wenn man sicherstellen kann, dass beide Verbindungsrichtungen die gleichen Eigenschaften aufweisen und sich die Verzögerung als Hälfte der Roundtriptime⁵ berechnet.

Als Parameter wurden bei ping die nachfolgend beschriebenen verwendet:

- c **<Anzahl>** Automatisch nach der definierten Anzahl von ICMP-Echo-Requests das Programm beenden
- i **<Intervall>** Wartezeit in Sekunden zwischen zwei ICMP-Echo-Requests
- q Nur die Auswertung am Ende ausgeben
- s **<Größe>** Paketgröße in Bytes (Nutzdaten)

4.2 ttcp / nuttcp und iperf

Die Tools ttcp[25], nuttcp[15] sowie iperf[7] bieten Optionen zur Bandbreitenmessung und Messung der Paketverlustrate. Sie können jeweils auf TCP und UDP-Basis arbeiten.

⁴Auf Layer 3 transparente Tunnel können natürlich nicht erkannt werden

⁵Abzüglich einer eventuell vorhandenen Verarbeitungszeit am gegenüberliegenden Messpunkt

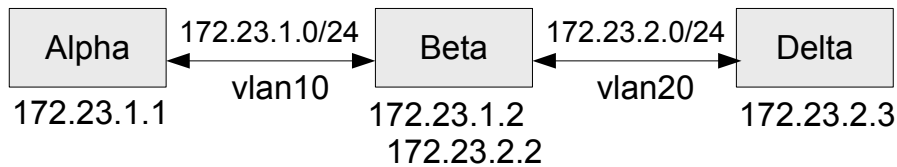


Abbildung 1: Testaufbau

Eine maximale zu sendende Bandbreite kann vorgegeben werden, welche bei UDP auch der gesendeten Datenrate entspricht. Die gesendeten Paketgrößen sind auch konfigurierbar. Bei nuttcp sind sowohl die Paketgrößen als auch die Bandbreitenangaben stets auf die übertragenen Nutzdaten bezogen, um die auf Layer 2 und Layer 3 genutzte Bandbreite zu berechnen sind noch die jeweiligen Header zu den Paketgrößen zu addieren.

Als Parameter wurden bei nuttcp die nachfolgend beschriebenen verwendet:

- R<Bandbreite>k Datenrate in KBit/s
- S Server Modus.
- T<Dauer> Zeitdauer in Sekunden eines Experiments
- l <Größe> Paketgröße in Bytes (Nutzdaten)
- u Verwende UDP

5 Testaufbau

Die für die Messungen auf Layer 2 und Layer 3 genutzte Hardware, Software und deren Konfigurationen wird im Folgenden vorgestellt.

5.1 Hardware und Netzwerksetup

In der Abbildung 1 ist das für die Messungen genutzte Testsetup abgebildet. Es besteht aus drei Rechnern welche mittels Fast Ethernet über einen Switch verbunden sind. Beta fungiert als Router und verfügt über zwei Netzwerkkarten. Über den Switch ist auch die Außenanbindung der Rechner realisiert, über die die Experimente gesteuert wurden.

Zwei identisch ausgestattete Rechner, 700 MHz AMD Duron mit 512 MB RAM, sind dabei Quelle (Alpha) und Senke (Delta) und als Router (Beta), auf dem das jeweilige Testscenario konfiguriert wird, dient ein AMD Athlon mit 1666 MHz und 1GB RAM.

Name	Version
Linux Distribution	Debian 5.0.3
Kernel	2.6.26-2-486 (alpha,delta) 2.6.26-2-686 (beta)
ping	iputils-sss20071127
nuttcp	5.5.5

Tabelle 1: Für die Messungen genutzte Software

In dem Netzwerk werden vLANs verwendet, um den Datenverkehr logisch noch weiter zu separieren. Mit den vLANs wird auch überprüft ob Pseudodevices⁶, wie die für vLANs verwendeten, problemlos mit den Link-Emulatoren verwendet werden können. Der Rechner „Alpha“ nutzt im vLAN 10 das Netzwerk 172.23.1.0/24 und hat die IP-Adresse 172.23.1.1. Der Rechner „Delta“ nutzt im vLAN 20 das Netzwerk 172.23.2.0/24 und hat die IP-Adresse 172.23.2.3. Der Rechner Beta ist Router zwischen den beiden Netzen. Er hat beide vLANs konfiguriert, und die IP-Adressen 172.23.1.2 und 172.23.1.2 für das jeweils entsprechende vLAN konfiguriert.

Es wurde ein Szenario aus mehreren Rechnern gewählt, um die Last der Emulationslösungen auf dem Router präziser bestimmen zu können. Es wurden auch bewusst leistungsschwache Rechner gewählt, um die Ressourcennutzung besser beobachten zu können. Bei virtuellen Netzen innerhalb einer Virtualisierungslösung würde sich am Vorgehen nichts ändern. Es würden zum Beispiel bei Xen[1, 26] die „vif“-Devices verwendet, anstelle der vLAN Interfaces.

Die verwendeten Softwareversionen werden in Tabelle 1 aufgeführt.

5.2 Konfiguration der Netzwerkemulationssoftware

5.2.1 TC/NetEm

NetEm ist als eine „classful queuing discipline“ des Traffic Control Frameworks (TC) implementiert. Dementsprechend können alle Techniken des Traffic Control Frameworks zur Filterung genutzt werden. Das Linux Traffic Control Framework beherrscht Queuing nur für ausgehenden Verkehr, sodass bei eingehendem Verkehr ein Pseudodevice genutzt werden muss.

Die Konfiguration von NetEm erfolgt entweder direkt beim Erstellen der „queuing discipline“, indem die Parameter bei der „add“-Aktion mit übergeben werden, oder später

⁶Andere Pseudodevices werden bei den verschiedenen Virtualisierungslösungen genutzt.

durch eine Veränderung der Konfiguration mittels „change“-Aktion. Dabei wird die zuvor bestehende Konfiguration durch die als Parameter angegebene Konfiguration ersetzt.

```
tc qdisc add dev ifb0 root handle 1:0 netem
tc qdisc add dev ifb0 root handle 1:0 netem loss 0.5%
tc qdisc change dev ifb0 root handle 1:0 netem delay 20ms
tc qdisc change dev ifb0 root handle 1:0 netem delay 100ms 50ms \
    distribution normal
tc qdisc change dev ifb0 root handle 1:0 netem duplicate 1%
tc qdisc change dev ifb0 root handle 1:0 netem corrupt 0.1%
```

Mehrere Parameter können kombiniert werden, um mehrere Effekte mit einer Instanz von NetEm zu emulieren.

```
tc qdisc change dev ifb0 root handle 1:0 netem loss 0.05% \
    delay 20ms duplicate 1%
```

TC bietet keine spezielle Syntax, um auf MAC-Adressen zu filtern, aber mit dem „u32“ und „u16“ Filter ist dies möglich. Der u32 und der u16 Filter sind einfache Filter, die prüfen ob ein definiertes Bitmuster im Header an der angegebenen Stelle vorliegt. Im folgenden Beispiel werden Pakete die von der MAC-Adresse „00:01:02:df:4d:db“ stammen und über das Pseudodevice vlan10 kommen auf das Pseudodevice „ifb0“ umgeleitet⁷. Dabei wird die 48 Bit MAC-Adresse in einen 16 Bit und einen 32 Bit Teil zerlegt, und mit den u16 und u32 Filter geprüft. Da der Ethernet-Header noch vor den Header der Nutzdaten (in dem Fall IP) kommt, muss ein negativer Offset bei dem Ort auf dem die Filter prüfen sollen angegeben werden.

```
tc filter add dev vlan10 parent ffff: protocol ip u32 \
    match u16 0x4ddb 0xFFFF at -4 \
    match u32 0x000102df 0xFFFFFFFF at -8 \
    flowid 1:1 action mirred egress redirect dev ifb0
```

Der u32 Filter verlangt die Angabe des Protokolls, von dem allerdings nur die Kennung im Ethernetheader bekannt sein muss, sodass bei neuen Protokollen statt der „sprechenden“ Kennung wie „ip“ auch direkt die Nummer angegeben werden kann.

Detaillierte Angaben zu der Konfiguration von NetEm finden sich in [17]; weitergehende Informationen zu TC im Allgemeinen sind in [16] zu finden.

⁷Um, wie zuvor beschrieben, NetEm als „queuing discipline“ zu nutzen.

5.2.2 Dummynet

Die Linuxportierung von Dummynet beinhaltet als Klassifizierer den Paketfilter ipfw. Die Konfiguration von Dummynet besteht deshalb auch aus zwei Teilen. Zum einen aus der Konfiguration der sogenannten Pipes, welche die Netzwerkemulation implementieren, und zum anderen aus der Konfiguration der Filter, welche die Pakete auf die Pipes verteilen.

Die Pipes können beim Anlegen parametrisiert werden und zur Laufzeit geändert werden. Pipes werden implizit angelegt, durch die erste Konfiguration.

```
ipfw pipe 4 config delay 0ms
ipfw pipe 4 config delay 20ms
ipfw pipe 4 config bw 2000Kbit/s queue 100
ipfw pipe 4 config plr 0.005
```

Die Verlustrate „plr“ wird als Anteil angegeben – nicht als Prozentangabe. Eine Verlustrate von 5% entspricht damit als Parameter 0.05. Mehrere Parameter für eine Pipe können kombiniert werden, um mit einer Pipe verschiedene Parameter zu emulieren.

```
ipfw pipe 4 config delay 20ms plr 0.0005
```

Die Verteilung auf die einzelnen Pipes übernimmt der Klassifizierer in ipfw. Neben der einfachen Filterung (z.B. nach Quelladresse, Interface o. Ä.) beherrscht ipfw zusätzlich die Funktion, Pakete die eine Regel erfüllen nur zu einem Anteil in die angegebene Pipe weiterzuleiten.

```
ipfw add pipe 4 dst-ip 172.23.1.0/24 out
ipfw add pipe 4 src-ether 00:01:02:df:4d:db in
ipfw add prob 0.5556 pipe 4 src-ip 172.23.1.0/24 out
```

Detaillierte Angaben zur der Konfiguration finden sich in [28].

5.3 Messungen

Die verschiedenen Parameter, welche die beiden verglichenen Tools TC/NetEm und Dummynet bereitstellen wurden jeweils getrennt analysiert. Das detaillierte Setup ist für die verschiedenen betrachteten Werte detailliert beschrieben.

5.3.1 Datenrate/Bandwidth

Zur Messung der Datenrate wird das Tool `nuttcp` und `ping` genutzt. Mit `nuttcp` wird für verschiedene Paketgrößen⁸, welche von relativ kleinen Paketen, wie sie etwa bei Voice over IP genutzt werden, bis zu großen Paketen reichen, die bei normaler Datenübertragung verwendet werden, jeweils UDP-Ströme mit einer Bandbreite von 90% bis 110% der konfigurierten Bandbreite⁹ gesendet. Das Skript, welches die Messungen durchführt, ist in dem Anhang A.1 abgedruckt. Zur genauen Bestimmung der erzielten Bandbreite wird bei der ersten Messung mit Paketverlusten, anhand der Anzahl der übertragenden Pakete, die erzielte Bandbreite (BW) berechnet. Dazu wird die Anzahl der Pakete (A) mit der konfigurierten Größe der Nutzdaten (S_N) zuzüglich der jeweils zutreffenden Header¹⁰ (UDP S_{UDP} , IP S_{IP} und Ethernet S_{Ether} ¹¹) multipliziert und durch die Dauer des Experiments (t) dividiert.

$$BW_{Dummysnet} = \frac{A * (S_N + S_{UDP} + S_{IP})}{t}$$

$$B_{TC-NetEm} = \frac{A * (S_N + S_{UDP} + S_{IP} + S_{Ether})}{t}$$

Die Abweichung von dem Sollwert wird dann prozentual berechnet:

$$Abweichung = (BW_{Ist} - BW_{Soll}) / BW_{Soll} * 100$$

Mit einer Reihe von ICMP-Echo Requests („ping“) wird bei unterschiedlichen Paketgrößen¹² zusätzlich die Round-Trip-Time ermittelt. Da die Bandbreitenbegrenzung in beide Richtungen konfiguriert ist und somit die Verbindung in beide Richtungen identische Eigenschaften aufweist, kann die Verzögerung für eine Richtung bestimmt werden, indem die gemessene Round-Trip-Time durch 2 dividiert wird. Die so ermittelte Verzögerung wird mit der aufgrund der Bandbreite und Paketgröße erwarteten Verzögerung verglichen.

⁸Nutzdaten: 128, 256, 512, 1024, 1448 Byte

Ethernetframelänge: 174, 302, 558, 1070, 1494 Byte

⁹66, 131, 262, 524, 1049, 2097, 4194, 8389, 16777 KBit/s

¹⁰Wie in Kapitel 7.1 beschrieben, wird bei Dummysnet zur Bandbreitenberechnung nur die Länge des Layer 3 Packetes herangezogen.

¹¹In die Größe des Ethernetheaders sind auch die 4 Bytes der Prüfsumme einberechnet

¹²Nutzdaten: 128, 256, 512, 1024, 1448 Byte

Ethernetframelänge: 170, 298, 554, 1066, 1490 Byte

5.3.2 Verzögerung/Delay

Zur Bestimmung der Verzögerung werden ICMP-Echo Requests („ping“) genutzt. Es wird zunächst ein Referenzwert für den Testaufbau ohne aktivierte Netzwerkemulation für die verwendeten Paketgrößen bestimmt.

Mit dem in Anhang A.2 abgedruckten Skript werden dann sowohl in NetEm als auch in DummyNet verschiedene Verzögerungen für eine Richtung konfiguriert und jeweils bei 10.000 Paketen bei jeder Paketgröße die RTT gemessen. Von der gemittelten RTT wird die Verzögerung, welche als Referenzwert bestimmt wurde, abgezogen, um die Verzögerung, welche durch die Netzwerkemulation erzeugt wird, präzise bestimmen zu können.

5.3.3 Paketverlustrate/Loss

Die Paketverlustraten wurden für zwei Szenarien erfasst.

Referenzwert: Um Referenzwerte für den Paketverlust zu erhalten, werden mit nuttcp Datenströme mit einer Bandbreite von 0,5 MBit/s bis 100 MBit/s¹³ bei drei unterschiedlichen Paketgrößen¹⁴ generiert. Die dabei jeweils aufgetretenen Paketverluste werden aufgezeichnet. Die zu testenden Konfigurationen sind:

- Ohne Netzwerkemulation
- TC/NetEm, Bandbreitenlimitierung von 100 MBit/s, Filterung auf Layer 2
- TC/NetEm, Bandbreitenlimitierung von 100 MBit/s, Filterung auf Layer 3
- DummyNet, Bandbreitenlimitierung von 100 MBit/s, Filterung auf Layer 2
- DummyNet, Bandbreitenlimitierung von 100 MBit/s, Filterung auf Layer 3

Für DummyNet mit der Filterung auf Layer 2 ist exemplarisch das verwendete Skript im Anhang A.3.1 abgedruckt. Das Skript zur Auswertung der Logdateien ist ebenfalls im Anhang A.3.1 abgedruckt.

Zielwert: Wie genau Vorgaben zum prozentualen Paketverlust von NetEm und DummyNet eingehalten werden, wird mit kleinen Paketen¹⁵ bei Bandbreiten von 1 MBit/s,

¹³in 0,5 Mbit/s Schritten erhöht

¹⁴Länge der Ethernetpakete 146, 796, 1494 Bytes, wieder um von kleinen bis großen Paketen das Verhalten zu repräsentieren.

¹⁵146 Bytes Länge des Ethernetframes

20 MBit/s, 40 MBit/s und 75 MBit/s gemessen. Die zu erzielenden Verlustraten sind 0,1%, 0,5%, 1%, 5% und 10%.

Die zu testenden Konfigurationen sind:

- TC/NetEm, Filterung auf Layer 2
- TC/NetEm, Filterung auf Layer 3
- Dummynet, Filterung auf Layer 2
- Dummynet, Filterung auf Layer 3
- Dummynet, Filterung auf Layer 2, Bandbreitenlimitierung von 100 MBit/s
- Dummynet, Filterung auf Layer 3, Bandbreitenlimitierung von 100 MBit/s

Das vollständige Skript der Messreihe ist in Anhang A.3.2 abgedruckt.

Zusätzlich wird für die Verlustraten bei verschiedenen Paketgrößen die Verlustraten per „ping“ mit einer Rate von 100 Paketen pro Sekunde und einem Stichprobenumfang von 10.000 Paketen gemessen.

5.3.4 Laufzeitschwankung/Jitter

Um die Laufzeitschwankungen zu messen, wird bei NetEm eine Verzögerung von 100 ms mit einer Streuung von ± 50 ms gemäß einer Normalverteilung konfiguriert.

Für Dummynet werden 11 Pipes konfiguriert, deren Eigenschaften nach einer Tabelle zur Normalverteilung parametrisiert werden, um sich einer Normalverteilung von 100ms ± 50 ms anzunähern. Eine Pipe mit einer bestimmten Verzögerung, die stellvertretend für ein Intervall um diesen Wert steht, wird jeweils mit einer dazu den Tabellenwerten entsprechenden Wahrscheinlichkeit gewählt. Dabei ist zu beachten, dass die bei ipfw anzugebenden Wahrscheinlichkeiten sich auf die verbleibenden, d.h. noch nicht von den zuvor definierten Regeln bearbeiteten Pakete, bezieht. Die genaue Konfiguration ist im Anhang A.4 abgedruckt.

Um den Jitter zu analysieren wird für unterschiedliche Paketgrößen¹⁶ jeweils eine Sequenz von 10.000 ICMP-Echo Requests („ping“) aufgezeichnet und die Verteilung der Verzögerungen mittels eines Skripts ausgewertet. Die Auswertung ergibt ein Histogramm für die erzielten Verzögerungen, bei der die Anzahl der empfangenen Pakete, bzw. die entsprechende Umrechnung in Prozent, für jede mögliche Verzögerung zwischen 0ms und

¹⁶Nutzdaten: 128, 256, 512, 1024, 1448 Byte
Ethernetframelänge: 170, 298, 554, 1066, 1490 Byte

200ms bestimmt wird. Sowohl das Messskript als auch das Auswertungsskript sind im Anhang A.4 abgedruckt.

5.3.5 Ressourcenbedarf

Zur Bestimmung des Ressourcenbedarfs von DummyNet und TC/NetEm wird mit dem Werkzeug „vmstat“ die Auslastung der CPU bei Bandbreiten von 1 bis 75 MBit/s gemessen. Als Referenzwert dient dabei die CPU-Auslastung ohne geladene Netzwerkemulation. Die konfigurierten Parameter sind für TC/NetEm und DummyNet identisch.

Um Einflüsse anderer Prozesse¹⁷ möglichst gering zu halten, wurde nur der Anteil der CPU-Auslastung betrachtet, die der Kernel selber nutzt („sys“). Dies ist ausreichend, da beide Link-Emulatoren vollständig im Kernel ablaufen.

Bei jeder Messung werden, 5 Sekunden nach dem beaufschlagen des Systems mit der zu testenden Bandbreite, 130 Sekunden lang sekundlich die aktuelle Prozessorauslastung aufgezeichnet. Die ersten 10 und letzten 20 Messwerte werden verworfen, um sicher zu gehen, dass die Werte im eingeschwungenen System gemessen sind. Von den verbleibenden 100 Messwerten wird ein Mittelwert gebildet.

Die Messungen finden mit einer Paketgröße von 146 Bytes (Framelänge Ethernet) statt. Diese relativ geringe Framelänge führt zu einer hohen Anzahl von Paketen pro Sekunde. Für DummyNet wurde zusätzlich eine Messung mit 1496 Bytes (Framelänge Ethernet) durchgeführt. Die verwendeten Skripte sind im Anhang A.5 abgedruckt.

Da vmstat nur Angaben in ganzen Prozentschritten macht, ist eine Abweichung nach oben der realen Last gegenüber der gemessenen Last von bis zu 1% möglich.

¹⁷Es wurde darauf geachtet, dass während der Testausführung keine CPU belastenden Programme liefen. Filtert den geringen Anteil, den die Programme zur Messung selbst benötigen, heraus.

Teil III

Implementierung

6 Anpassungen von Dummynet

6.1 Ausgangssituation

Für den Link Emulator Dummynet[28, 10] existiert ein Patch[19], der den Klassifizierer (ipfw) um Optionen und interne Strukturen erweitert, die Filterung auf Layer 2 nach MAC-Adressen ermöglichen. Der Patch ist allerdings nur für FreeBSD[27] verfügbar, sodass er nicht direkt auf den Linux[23] Patch[11] angewendet werden kann.

6.2 Anpassungen

Um den Layer 2 Patch[19] auf die Linuxportierung[11] von Dummynet zu adaptieren, waren eine Vielzahl von Anpassungen notwendig, von denen im Folgenden einige ausgewählt vorgestellt werden.

Andere Verzeichnisstruktur: Durch eine deutlich von der in FreeBSD abweichenden Verzeichnisstruktur der Linuxportierung, war bei der Anwendung des Layer 2 Patches stets herauszufinden, in welcher Datei die entsprechenden äquivalenten Datenstrukturen zu finden waren. Erschwert wurde dies zusätzlich dadurch, dass einige Deklarationen (in den „Header-Dateien“¹⁸) in der Linuxportierung redundant an zwei verschiedenen Orten vorhanden waren. Einmal für das Kernelmodul, einmal für das im Userspace laufende Konfigurationsinterface.

Includes und Symbole: Es mussten einige FreeBSD Deklarationen (eingebundene „Header“-Dateien) gegen die Linux-Äquivalente ausgetauscht werden. Für einige BSD-Spezifische Kernel-Symbole gibt es keine Entsprechungen im Linuxkernel, sodass diese manuell in eigene Headerdateien übernommen werden mussten.

¹⁸Dateien in denen Datenstrukturen definiert sind, und Signaturen von Methoden.

Datenstrukturen: Neben der Übernahme von einigen Datenstrukturdefinitionen, welche in den Linux-Headern nicht vorhanden waren, war die größte notwendige Änderung die Adaption auf die sich von der FreeBSD unterscheidenden Repräsentation der internen Datenstrukturen eines Pakets. Für Layer 3 war das bereits durch die vorhandene Linuxportierung[11] gemacht worden. Für Layer 2 war dies selber zu leisten.

In dem ursprünglichen FreeBSD-Patch wurde in die Kerneldatenstruktur ein optionaler Zeiger namens „eh“ eingeführt, welcher nur bei vorhandenen Layer 2 Adresse ungleich „NULL“ ist. In der Linuxportierung wird ein Teil der Felder mit den entsprechenden Werten belegt und die originale Linux-Datenstruktur als Zeiger „m“ weiterhin referenziert. Da die Ethernetadressen in den FreeBSD Kerneldatenstrukturen anders abgelegt werden als in den Datenstrukturen des Linuxkernels, konnte keine einfache Abbildung auf die FreeBSD-Datenstruktur vorgenommen werden. Daher wurde an sämtlichen Stellen an denen die „eh“ Datenstruktur verwendet wird, bei der Portierung auf Linux die Programmlogik so geändert, dass für die Layer 2 Adressen direkt auf die Linux-Datenstruktur zugegriffen wird. Die Prüfung, ob ein Layer 2 Header vorhanden ist, musste dementsprechend an vielen Stellen angepasst werden.

Ablauf: Der FreeBSD-Kernel bietet den Paketfiltern zwei unterschiedliche Schnittstellen, je nachdem ob die Pakete reine Layer 3 oder auch Layer 2 Informationen enthalten. Dies wird von dem ipfw Paketfilter genutzt, welcher der Klassifizierer für Dummysnet ist, um zu unterscheiden, ob es Pakete sind, welche auch auf Layer 2 Informationen geprüft werden müssen. In dem Linuxkernel stehen keine derartig differenzierten Schnittstellen zur Verfügung¹⁹, sodass eine „Heuristik“ implementiert werden musste, die anhand der belegten Zeiger in der Linux-Datenstruktur „m“²⁰ entscheidet ob Ethernet-Header, und damit Layer 2 Header vorhanden sind.

Der Linuxkernel bietet, wie in Abbildung 2 dargestellt, ausgehend vom „BROUTING“-Hook auch eine Schnittstelle, die nur Pakete liefert, welche über das Bridge Subsystem transportiert werden und damit sicher einen Layer 2 Header haben. Dieser Hook ist in Abbildung 2 durch hellgraue Pfeile mit dem Dummysnet-Modul verbunden. Diese wurde nicht genutzt, da der Implementierungsaufwand als zu hoch beurteilt wurde und dies nicht für den, in dieser Arbeit gewählten, Vergleich der Messungen auf Layer 2 und

¹⁹Der Linuxkernel stellt im Rahmen des netfilter Framework auch den sogenannten „BROUTING“-Hook zur Verfügung, mit dem Pakete untersucht werden können, die nur gebridget werden. In der aktuellen Implementierung der Linux-Portierung werden die so genannten PRE- und POST-Routing Hooks verwendet. Diese setzen voraus, dass das Paket geroutet und nicht nur gebridget wird.

²⁰Genauer ob der Zeiger `m->m_skb->mac_header` ungleich „NULL“ ist.

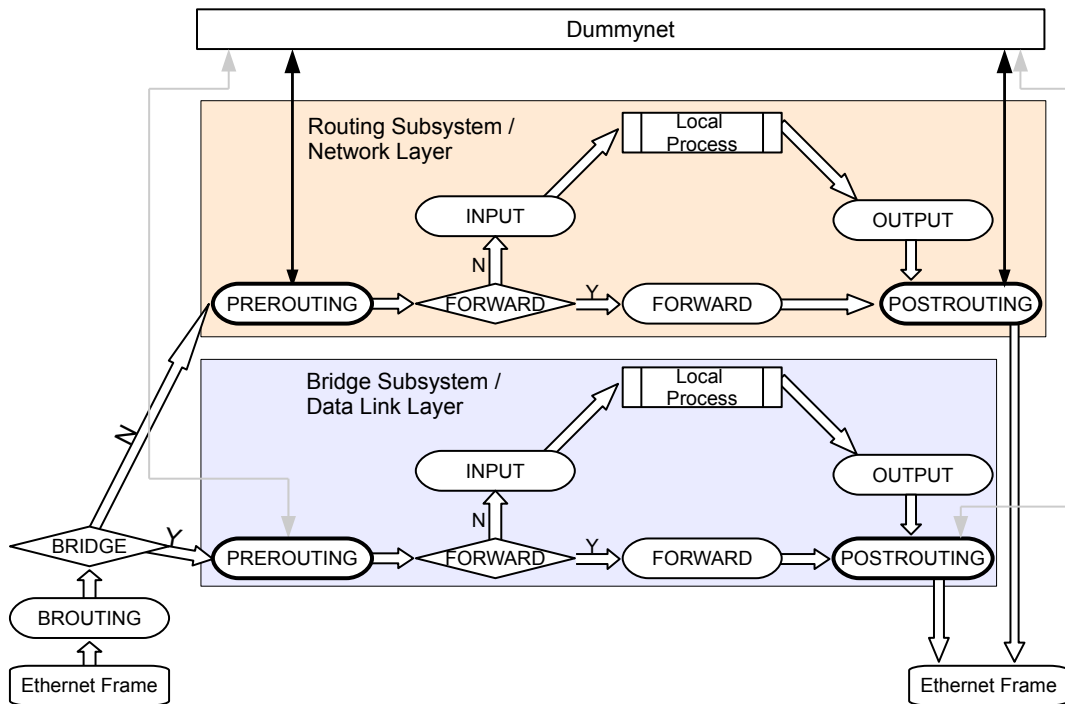


Abbildung 2: Netfilter-Hooks im Linuxkernel

Layer 3 relevant ist²¹. Die „Heuristik“ wäre auch bei der Nutzung der Hooks in dem Bridge Subsystem notwendig gewesen, um Pakete nach Layer 2 Header bearbeiten zu können, die im Routing Subsystem weitergeleitet werden.

Teil IV

Tests

7 Erfahrungen

7.1 Unterschiedliche Bandbreitenberechnungen

Die unterschiedlichen Werkzeuge haben, für die der Bandbreitenberechnung zugrundeliegenden Paketgröße, unterschiedliche Definitionen, was zu einem Paket gehört. Daher muss zwischen den unterschiedlichen Definitionen umgerechnet werden.

²¹Um die Messungen vergleichen zu können, muss der Rechner „Beta“ in beiden Szenarien als Router arbeiten, sodass der in jedem Fall der Hook im Routing-Subsystem verwendet würde.

Die experimentell ermittelten Größen werden im Folgenden vorgestellt:

TC/NetEm: Die Bandbreitenlimitierung der genutzten „Token Buffer“ Implementierung im TC-Framework nutzt zur Berechnung der Bandbreite die Länge des Layer 2/Ethernet-Paketes. Dies beinhaltet für die Messungen den Ethernet-Header und die CRC-Prüfsumme, jedoch nicht die Präambel.

Dummynet: Für die Bandbreitenlimitierung geht Dummynet von der Länge des Layer 3/IP-Paketes aus. Das heißt, dass der Ethernetheader bei der Berechnung nicht berücksichtigt wird.

nuttcp: Die Bandbreitenangaben und Limitierungen von nuttcp sind auf die Nutzdaten bezogen. Das heißt, dass weder TCP/UDP Header eingerechnet ist, noch der IP-Header oder der Ethernet-Header.

ping: Bei Ping werden ebenfalls rein die „Nutzdaten“ als Paketgröße angegeben, ICMP Header, IP-Header und Ethernet-Header werden dabei nicht berücksichtigt. Bei der Ausgabe wird der ICMP-Header mit einberechnet.

7.2 Einschränkungen / Fehlfunktionen

Der Layer 2 Patch[19] hat einen Bug eingeführt, der bei hoher bidirektionaler Auslastung der Netzwerkverbindung (ab 80 MBit/s bei 100 MBit/s Verbindungen beobachtet) zu einer Kernelpanic während der Interrupt-Behandlung führen kann. Bei der Verwendung zweier Netzwerkkarten, welche jeweils nur ein- oder ausgehenden Verkehr haben, wie für das Testsetup beschrieben, konnte der Fehler nicht reproduziert werden. Da dieser Fehler nicht näher eingegrenzt werden konnte²², wurde er nicht weiter verfolgt.

Dummynet hat ein im Quelltext vorgegebenes Limit für die Puffergröße bei der Bandbreitenlimitierung von maximal 100 Paketen. Dies begrenzt, da Dummynet bei der Bandbreitenlimitierung aktuell eine Verzögerung von ca. 3ms hat, den maximalen Durchsatz von Paketen pro Sekunde. Dies wird in Kapitel 8.1.3 näher ausgeführt.

Der verwendete „POST_ROUTING“ Hook für ausgehende Netzwerkpakete greift offensichtlich vor der Zuordnung der Ziel-MAC-Adresse. Daher kann bei ausgehenden Paketen

²²Es war auch keine Abhängigkeit von der Anzahl der Pakete/s, der Interrupts/s oder der CPU-Auslastung festzustellen

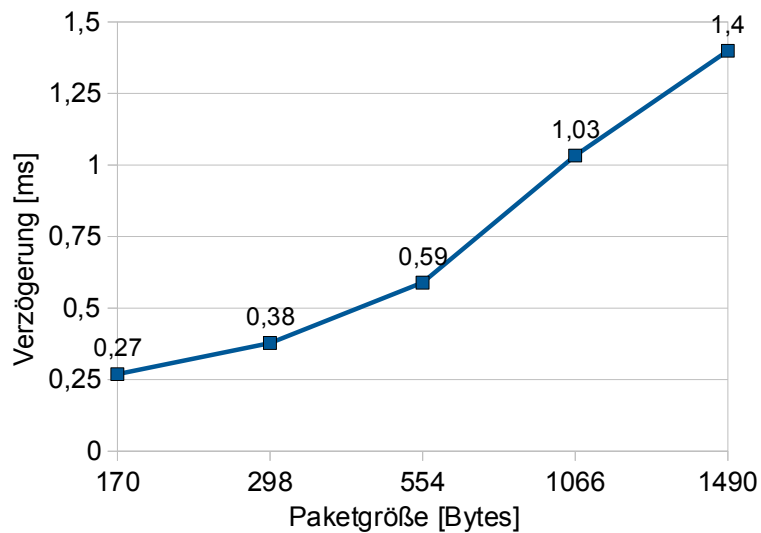


Abbildung 3: Referenzwerte für die Verzögerung bei verschiedenen Paketgrößen

nicht nach der Ziel-MAC-Adresse gefiltert werden. In Szenarien wo diese Funktionalität benötigt wird, müsste das Modul noch an den BROUTING-Hook angepasst werden, dies wird weiter in Kapitel 9.2 dargestellt.

8 Auswertung der Messergebnisse

Für die im Kapitel 5.3 beschriebenen Messreihen werden im Folgenden einige ausgewählte Auswertungen dargestellt und erläutert.

8.1 Referenzwerte

Als Vergleichsbasis für die weiteren Messungen wurden Verzögerung bei verschiedener Paketgröße und Paketverluste bei verschiedenen Bandbreiten gemessen. Das genaue Vorgehen ist in den Kapiteln 5.3.2 und 5.3.3 dokumentiert.

8.1.1 Basissystem

In Abbildung 3 ist die mittlere Verzögerung für verschiedene Paketgrößen abgebildet. Bei diesem Diagramm sind keine Auffälligkeiten festzustellen. Die Verzögerungen steigen wie zu erwarten mit der Paketgröße an.

In Abbildung 4 ist der Paketverlust für die Bandbreiten von 0,5 MBit/s bis 100 MBit/s in

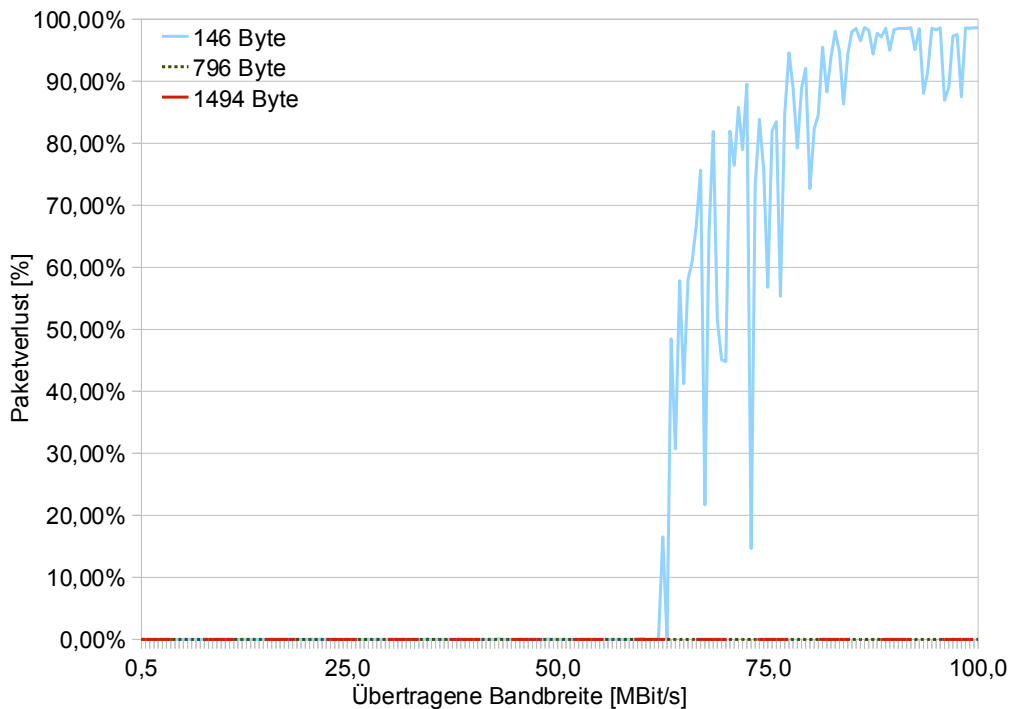


Abbildung 4: Referenzwerte für den Paketverlust in Prozent ohne Emulationstechniken bei verschiedenen Bandbreiten

0,5 MBit/s Schritten abgebildet. Auffallend ist, dass bei den 142 Byte²³ langen Paketen ab etwa 63 MBit/s, was etwa 54.350 Pakete/s entspricht ein steiler Anstieg der Paketverlustrate von 0% auf etwa 50% stattfindet, der im Folgenden auf nahezu 100% steigt. Dies dürfte der Bereich sein, in dem der Router aufgrund der begrenzten Ressourcen und der, mit der Paketanzahl verbundenen, Zahl von Interrupts in den Überlastbereich gerät und beginnt den Inhalt von Puffern vollständig zu verwerfen.

8.1.2 TC/NetEm

Die Paketverlustraten für TC/NetEm verlaufen, wie in Abbildung 5 zu erkennen, im Grundsatz ähnlich zu den Kurven des Basissystems. Ab etwa 64 MBit/s steigt die Paketverlustrate für die 142 Byte langen Pakete stark an. Im Unterschied zu dem Basissystem ist der Anstieg jedoch langsamer und der Paketverlust stabilisiert sich mit ca. 35% auf einem niedrigeren Niveau. Dies lässt sich mit anderen verwendeten Queuing-Strategien erklären, bei denen die Puffer nicht vollständig verworfen werden.

Es ist kein signifikanter Unterschied zwischen der Filterung auf Layer 2 und auf Layer 3

²³100 Byte Nutzdaten, 142 Byte als Ethernetpaket

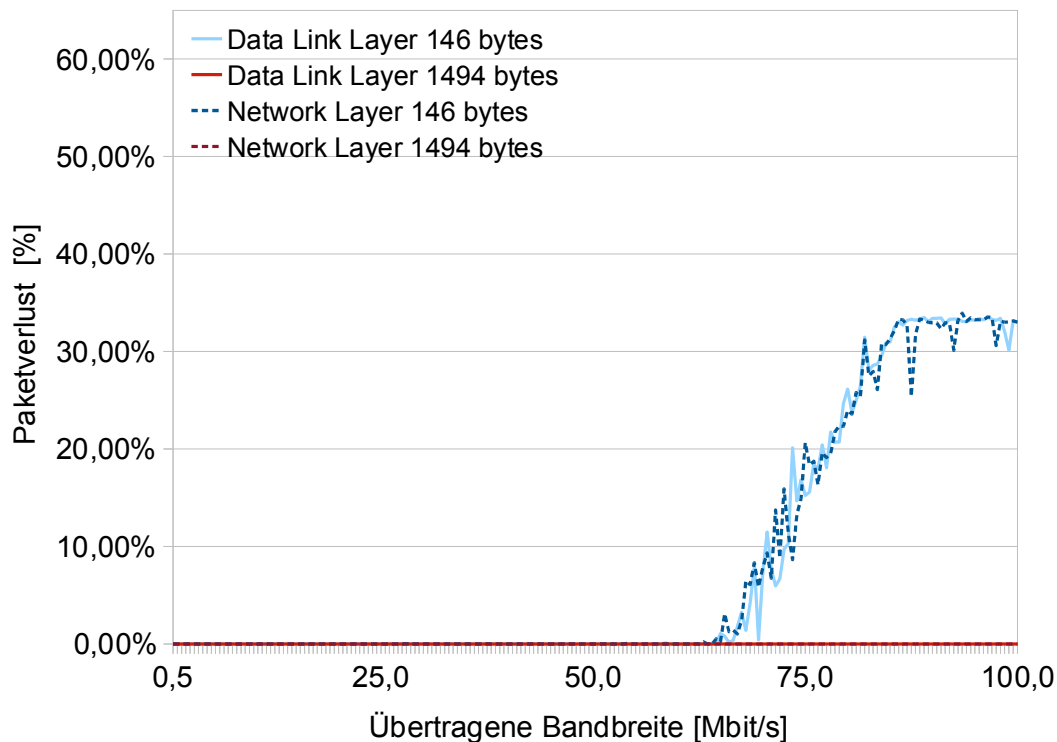


Abbildung 5: Paketverlust in Prozent bei Konfiguration von TC/NetEm auf 100 MBit/s festzustellen.

8.1.3 Dummynet

Die Paketverlustraten für Dummynet sind in Abbildung 6 abgebildet. Auffällig ist, dass ein Paketverlust bei unter 30 MBit/s beginnt und annähernd linear auf 62% bei 97,5 MBit/s ansteigt. Die Paketverlustrate war bei der Filterung auf dem Data Link Layer bei hohen Bandbreiten etwas erhöht, aber in einer vergleichbaren Größenordnung.

Dieser Paketverlust lässt sich erklären. Bei der konfigurierten Bandbreitenbegrenzung konnte eine Verzögerung von etwa 3ms beobachtet werden. Für Dummynet lässt sich eine maximale Queuelänge von 100 Paketen konfigurieren. Daher folgt nach Little's Law[24]

$$L = \lambda * W$$

mit der (maximalen) „mittleren“ Anzahl von Nachrichten im System $L = 100$ und der Verweilzeit (Delay) $W = 0,003s$ als maximale Ankunftsrate ohne Verluste:

$$\lambda = \frac{L}{W} = \frac{100 \text{ Pakete}}{0,003s} = 33.333.\bar{3} \frac{\text{Pakete}}{s}$$

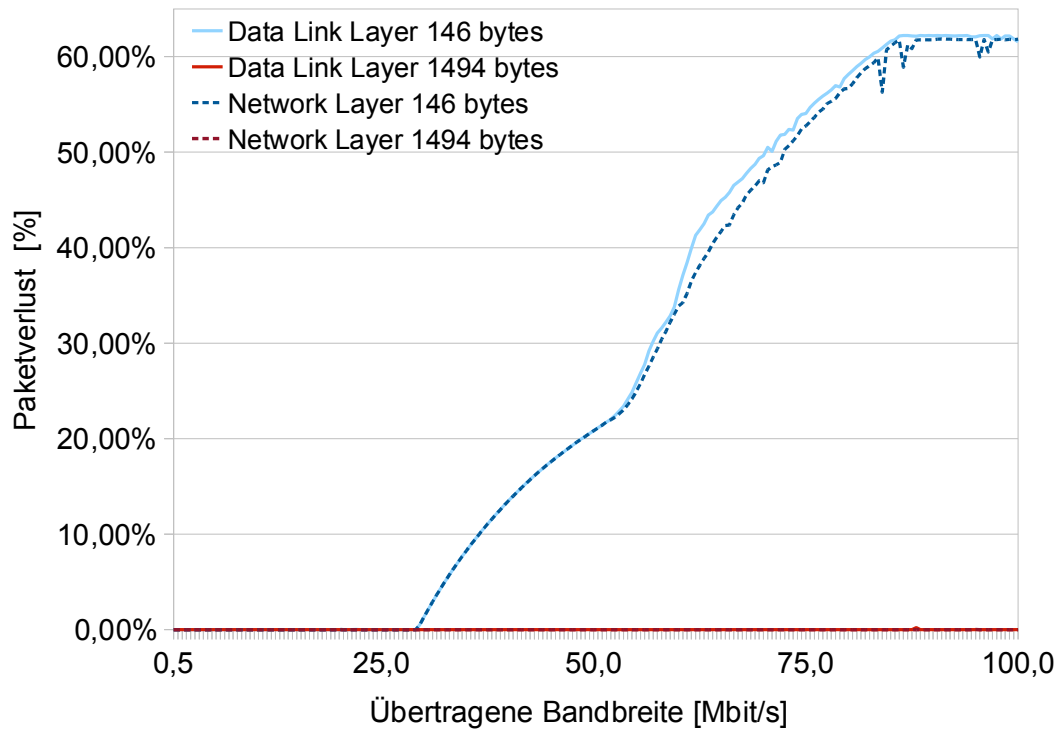


Abbildung 6: Paketverlust in Prozent bei Konfiguration von Dummynet auf 100 MBit/s

Die ersten Paketverluste wurden bei 29,5 MBit/s beobachtet. Dies entspricht, bei der gewählten Paketgröße, etwa $25.250 \frac{\text{Pakete}}{\text{s}}$. Daraus ergibt sich nach Little's Law eine mittlere Warteschlangenlänge von $L = 25.250 \frac{\text{Pakete}}{\text{s}} * 0,003\text{s} = 75,75 \text{Pakete}$. Da die Ankunftsrate aufgrund verschiedenster Puffermechanismen²⁴ nicht vollkommen konstant ist, ist dies ein durchaus plausibler Wert.

8.1.4 Schlussfolgerung

Um für die Messungen der Funktionsweise der Link-Emulatoren nicht in Bereiche zu gelangen in denen Überlastverhalten die Messergebnisse verfälscht, wurden für die Messungen in Kapitel 8.3, Kapitel 8.5 und Kapitel 8.6 Bandbreiten unterhalb von 20 MBit/s gewählt. Für das Kapitel 8.2 und Kapitel 8.4 wird die Bandbreite auf 75 MBit/s limitiert, um auch die Auswirkungen des Überlastverhaltens zu analysieren.

8.2 CPU Last der verschiedenen Techniken

Für die in Kapitel 5.3.5 beschriebene Konfiguration ist in Abbildung 7 der CPU-Res-

²⁴z.B. Netzwerkkarten die nur ein Interrupt für mehrere Pakete auslösen

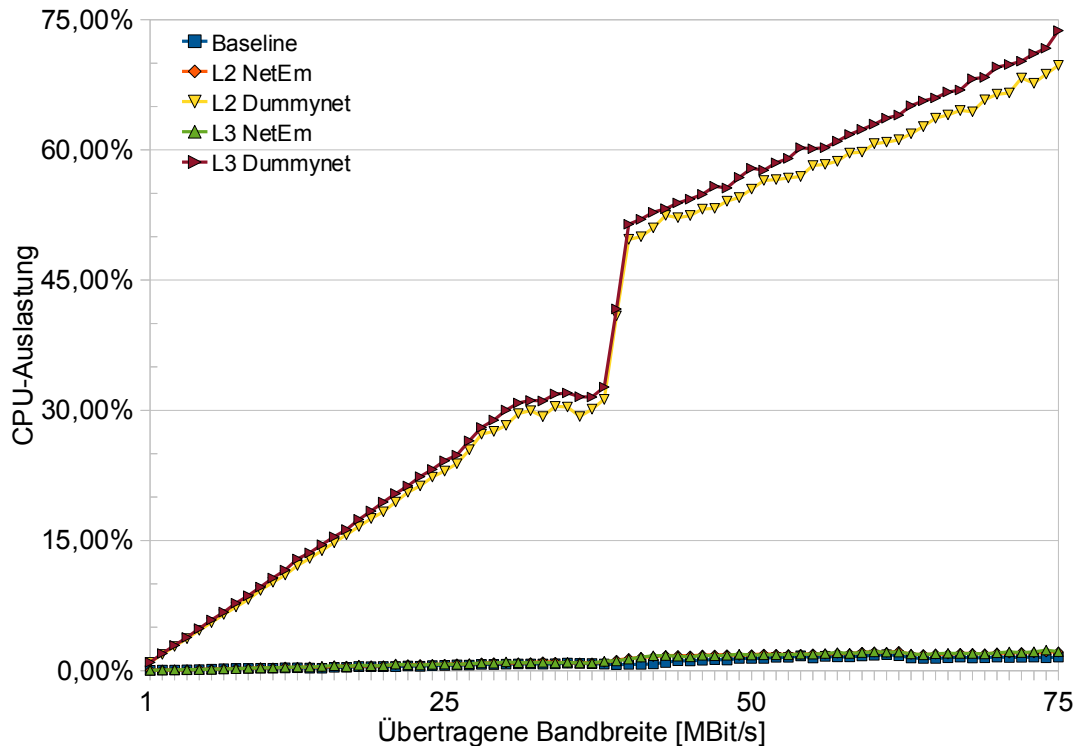


Abbildung 7: CPU-Nutzung in Prozent bei geladenen und auf 100 MBit/s konfigurierten Dummynet für verschiedenen Bandbreiten

sourcesbedarf²⁵ auf den Router Beta für verschiedene Bandbreiten aufgetragen. Gut zu erkennen ist, dass der Ressourcenbedarf für die TC/NetEm Lösung sich nicht signifikant von der Basiskonfiguration unterscheidet und es auch keinen erkennbaren Unterschied gibt, ob auf Layer 2 oder Layer 3 gefiltert wird.

Wenn man sich die Werte im Detail ansieht, kann man erkennen, dass Ressourcenbedarf für die Filterung auf Layer 2 etwa 10-30% über den Werten des Basis-Setup liegt und für die Filterung auf Layer 3 bei etwa 15-30% mehr Ressourcen verwendet werden müssen als im Basis-Setup. Dies ist in absoluten Zahlen aber eine Differenz von unter 1%, und damit im Rahmen der Messungenauigkeit von ca. 1%.

Die CPU-Last für Dummynet weicht deutlich von der Basiskonfiguration ab. Sie ist zum einen signifikant höher – bei 10 MBit/s beträgt sie mit 9,26% mehr als das 30 fache des Basis-Setups, welches 0,28% der CPU-Zeit beansprucht. Auffallend ist auch für beide Konfigurationen mit Dummynet der Sprung von etwa 30% CPU-Auslastung bei 38 MBit/s auf etwa 50% CPU-Auslastung bei 40 MBit/s. Auffällig ist, dass diese Ände-

²⁵An der Stelle sei noch einmal daran erinnert, dass aufgrund der verwendeten Messtechnik die tatsächliche CPU-Auslastung bis zu 1% höher sein kann.

zung offensichtlich keine Änderung in den Paketverlusten an der Stelle bewirkt (wie in Kapitel 8.1 zu erkennen). Da bei 39 MBit/s eine Paketrate von etwa $33.400 \frac{\text{Pakete}}{\text{Sekunde}}$ erreicht wird, die nach der Berechnung in Kapitel 8.1.3 für eine ständig vollständig gefüllte Warteschlange notwendig ist, könnte eine ineffiziente Behandlung von Paketankünften bei vollständig gefüllter Warteschlange für den großen Zuwachs zwischen 38 MBit/s und 40 MBit/s verantwortlich sein.

Der Layer 2 Patch hat den Ressourcenbedarf im Vergleich zur Layer 3 Filterung um etwa 10% erhöht, dies kann man besonders gut in dem Bereich ab 40 MBit/s erkennen. Dies ist auch damit zu erklären, dass bei der Implementierung zusätzlich vorher nicht vorhandene Codepfade hinzugekommen sind, deren Bedingungen überprüft werden müssen.

8.3 Genauigkeit der Bandbreitenlimitierung

Bei der Messung der Bandbreiten gab es keine nennenswerten Auffälligkeiten. Exemplarisch ist für die Bandbreite 8389 KBit/s in Abbildung 8 die erreichte Abweichung von der Bandbreite dargestellt, wenn der Sender versucht mit 101% der Bandbreite zu senden. Die erreichte Bandbreite wurde anhand der in Kapitel 5.3.1 vorgestellten Formel mittels der gesendeten Pakete abzüglich der verlorenen Pakete, im jeweiligen Testzeitraum, berechnet.

Paketverluste bei kleinen Paketgrößen können gut damit erklärt werden, dass aufgrund von unterschiedlichen Einflussfaktoren (nuttcp sendet mehrere Pakete zeitnah zusammenhängend, Puffer im Kernel, Netzwerkkarte fasst mehrere Interrupts zusammen u. Ä.) der Verkehr leicht burstartig verlaufen kann und bei der hohen Anzahl von Paketen dann relativ schnell die Puffer überlaufen. Die leichte Bandbreitenüberschreitung bei großen Paketgrößen lässt sich damit erklären, dass am Ende der Messdauer noch die Puffer im Router leerlaufen. Diese beinhalten bei der konfigurierten Bandbreite bis zu 33 Pakete.

In Abbildung 9 ist für verschiedene konfigurierte Bandbreiten bei zwei exemplarischen Paketgrößen die jeweils gemessene Verzögerung und die theoretisch bei einer „echten“ Verbindung dieser Bandbreite zu erwartende Verzögerung abgebildet. Die zu erwartende Verzögerung V berechnet sich aus der Paketgröße S und der Bandbreite BW :

$$V = \frac{S}{BW}$$

Gut zu erkennen ist, dass TC/NetEm diese Verzögerung nicht berücksichtigt. Die gemessene Verzögerung ist nahezu identisch mit der in Kapitel 3 gemessene Verzögerung

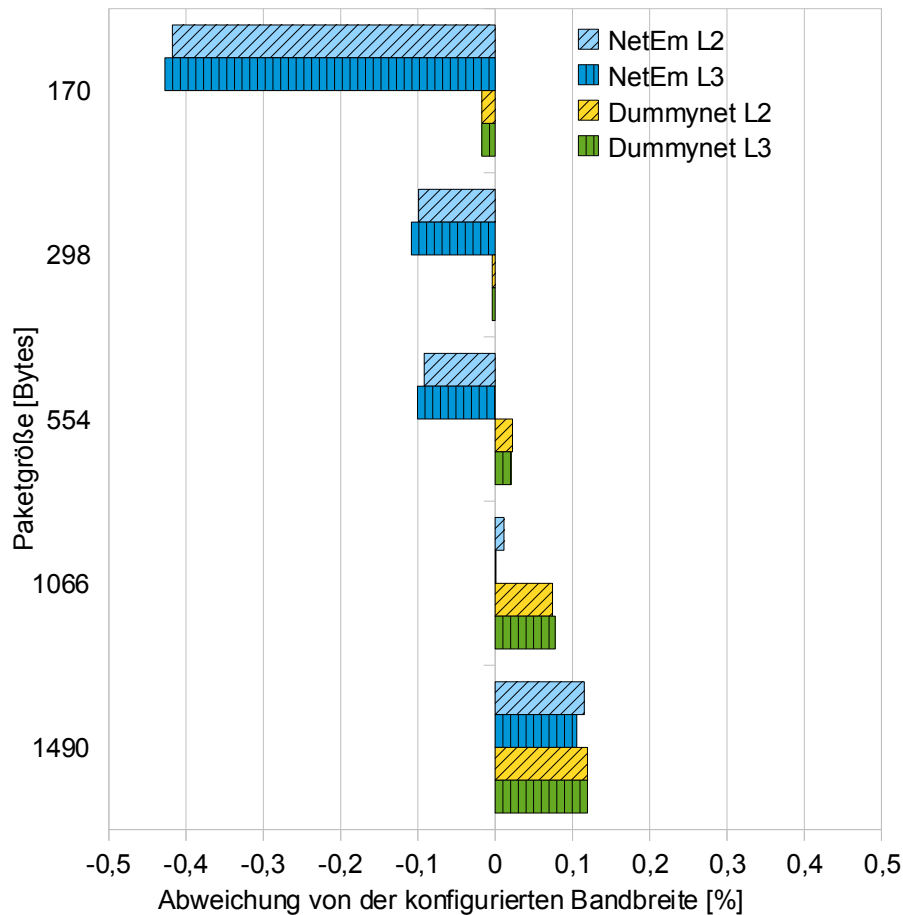


Abbildung 8: Abweichung von der konfigurierten Bandbreite von 8389 KBit/s bei verschiedenen Paketgrößen.

ohne Link-Emulator. DummyNet zeigt für die 170 Byte großen Pakete und die Bandbreiten von 64 KBit/s bis 256 KBit/s Messergebnisse, welche nennenswerte Ähnlichkeiten mit den erwarteten Werten aufweisen. Bei den 1500 Byte großen Paketen und höheren Bandbreiten tritt dieser Effekt aber nicht mehr auf und die Verzögerung stabilisiert sich bei etwa 3ms.

Da das Bandbreitenlatenzprodukt von der Bandbreite und der Paketgröße abhängig ist, kann es auch nicht durch eine zusätzlich konfigurierte feste Verzögerungen emuliert werden²⁶. Bei DummyNet kommt zusätzlich der Effekt hinzu, dass Verzögerungen unter 3ms nicht erzielt werden können.

²⁶Bei bekannter fester Paketgröße reicht doch ein festes Delay; oder es kann durch sehr viele unterschiedliche Delays, die abhängig von der Paketgröße konfiguriert werden, angenähert werden.

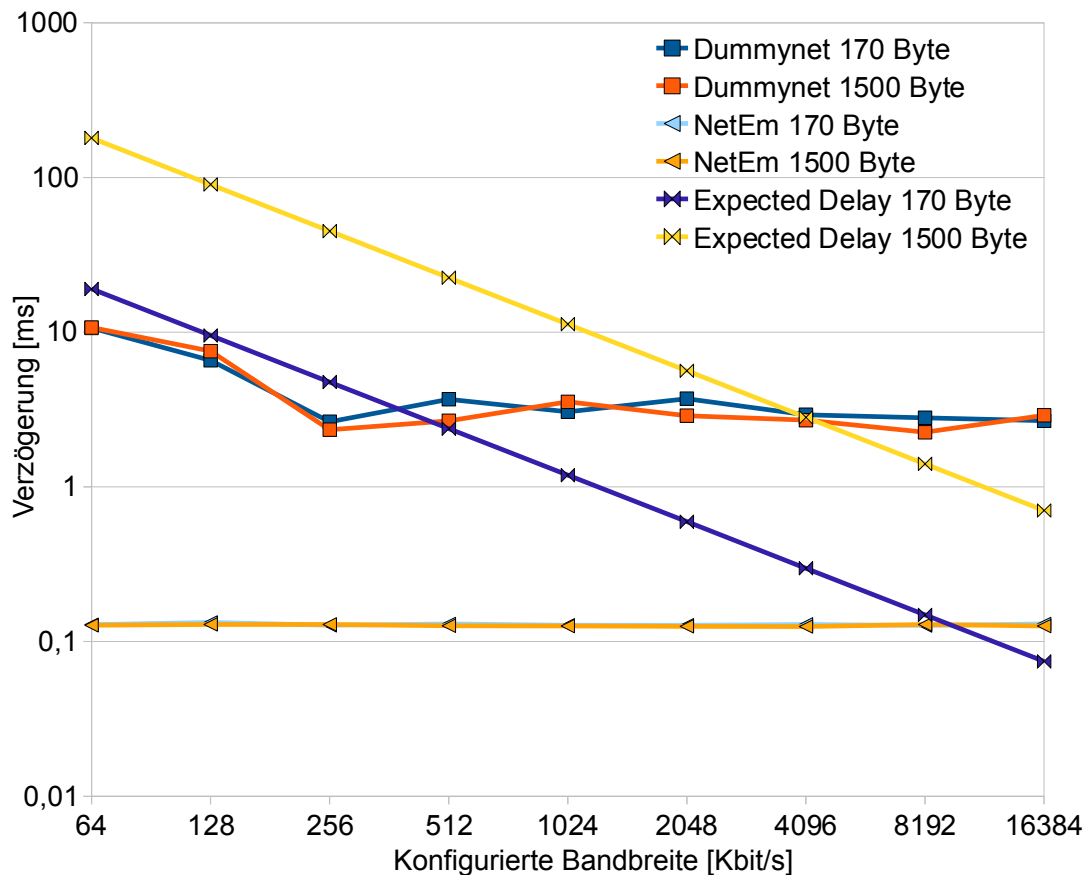


Abbildung 9: Vergleich der Latenz bei verschiedenen Paketgrößen und Bandbreiten mit dem theoretischen Bandbreitenlatenzprodukt

8.4 Genauigkeit der Paketverlustrate

Die Paketverlustraten bei geringer Bandbreite per „ping“ bewegen sich im Bereich der Erwartungen. Exemplarisch für die Paketlänge von 170 Bytes sind die Daten in Tabelle 2 verzeichnet. Für den relativ geringen Stichprobenumfang liegen die Werte gerade bei den geringeren Paketverlustraten sehr nahe am Soll. Bei den höheren Paketverlustraten (5% und 10%) sind hingegen schon deutliche Abweichungen festzustellen.

Wie in Abbildung 10 und Abbildung 11 zu erkennen ist, zeigt sich bei bei der Messung mit 1 MBit/s²⁷ ein ähnliches Bild. Bei den höheren Bandbreiten reduzierte sich die Streuung, sofern nicht durch Überlasteffekte der Paketverlust signifikant steigt. Auffallend ist, dass abweichend von der Messung in Kapitel 8.1.3 in Abbildung 10 bei 40 MBit/s noch keine relevante Abweichung von der konfigurierten Paketverlustrate aufge-

²⁷Das ergab in der Messdauer ca. 25.000 Samples, und damit mit der Stichprobe in der Tabelle vergleichbar.

Emulation	0,1%	0,5%	1%	5%	10%
Dummysnet L2	0,1%	0,47%	1,07%	5,29%	9,49%
Dummysnet L3	0,1%	0,55%	1,13%	4,98%	9,79%
TC/NetEm L2	0,08%	0,51%	0,98%	4,5%	9,84%
TC/NetEm L3	0,11%	0,4%	0,92%	5,2%	9,6%

Tabelle 2: Paketverlustraten bei einem Stichprobenumfang von 10.000 Paketen und verschiedenen Ziel-Paketverlustraten

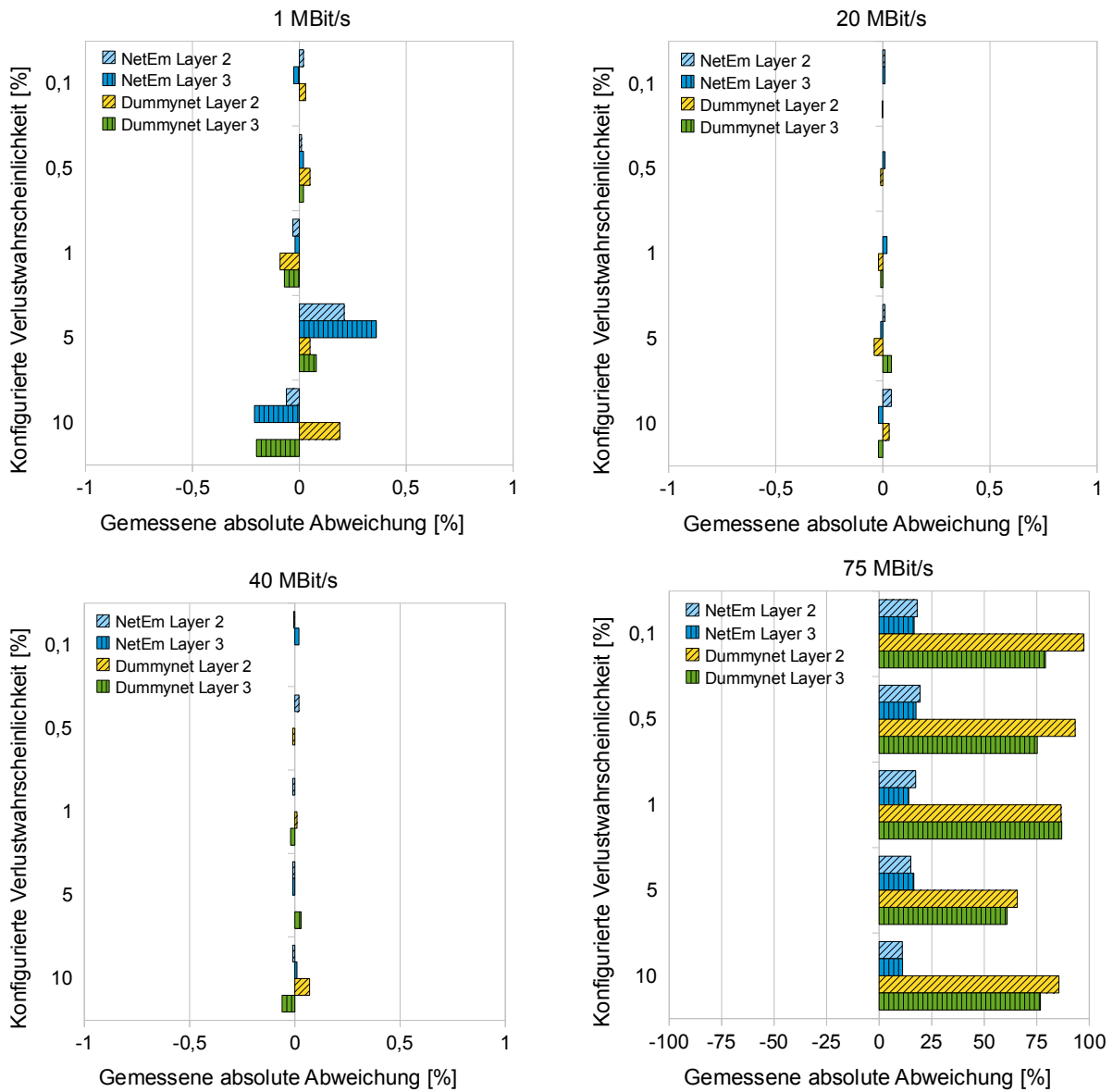


Abbildung 10: Paketverluste bei verschiedenen Bandbreiten und vorgegebenen Paketverlustraten

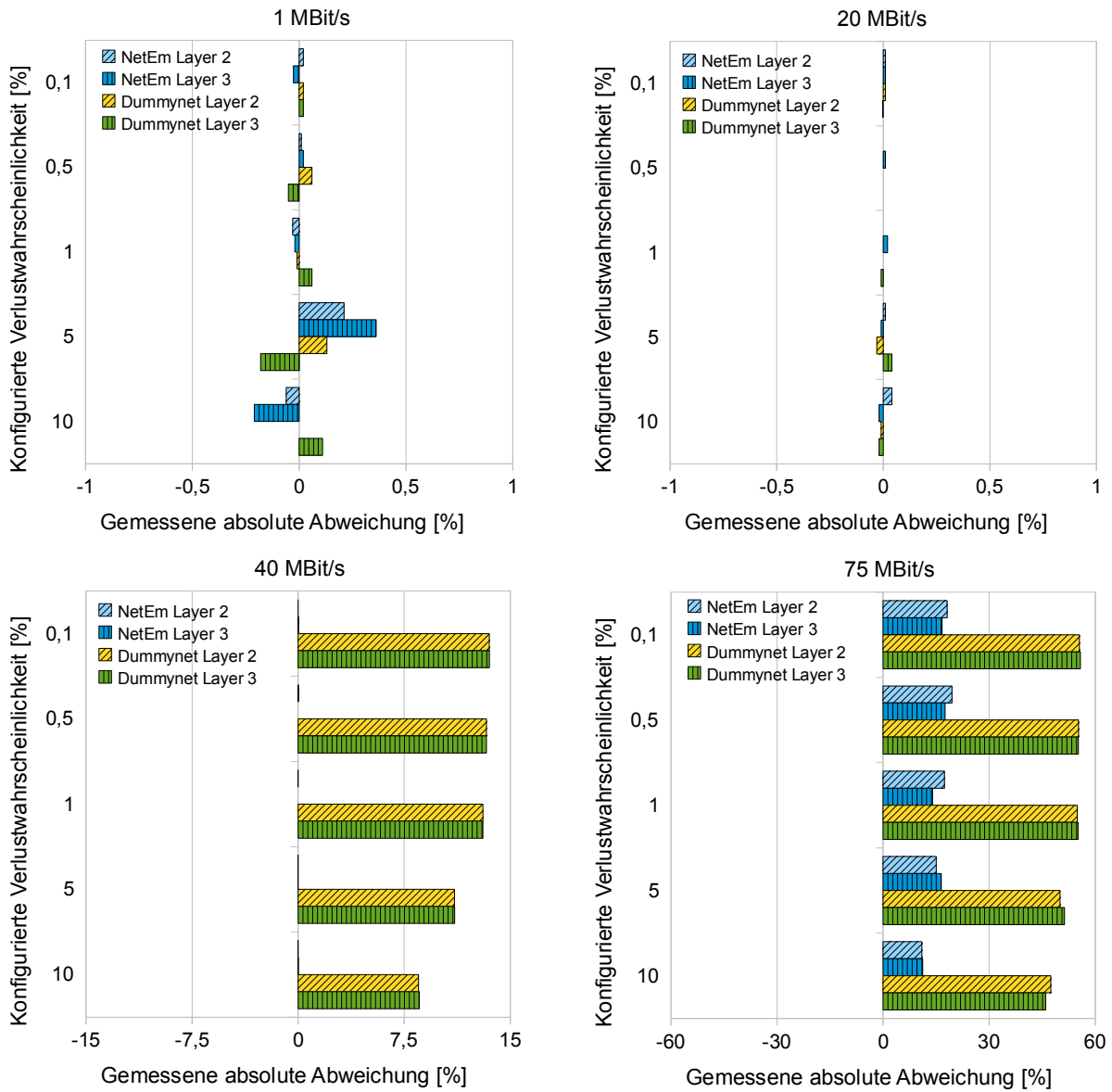


Abbildung 11: Paketverluste bei verschiedenen Bandbreiten und vorgegebenen Paketverlustraten. Dummynet mit einem Bandbreitenlimit von 100 MBit/s.

Emulation\Vorgabe	Verzögerung in ms							
	5	10	20	50	100	200	500	1000
Dummynet L2	5,97	11,69	18,4	50,84	98,24	198,4	498,25	998,35
Dummynet L3	5,93	11,69	18,62	50,54	98,2	198,41	498,26	998,216
TC/NetEm L2	5	10	20,01	50	100,01	200,01	500,01	1000,01
TC/NetEm L3	5	10	20,01	50,01	100,01	200,01	500,01	1000,01

Tabelle 3: Delay nach Herausrechnen der Verzögerung im System ohne Emulationssoftware für verschiedene vorgegebene Verzögerungen bei Paketen von 170 Bytes

treten ist. Bei einer konfigurierten Bandbreitenbegrenzung ist – wie in Abbildung 11 zu sehen – eine ähnliche Verlustrate zu erkennen wie in Kapitel 8.1.3. Daraus kann man schlussfolgern, dass für die hohe Verlustrate bei Dummynet der Codeteil verantwortlich ist, der die Bandbreitenlimitierung implementiert.

8.5 Genauigkeit der Verzögerung

In der Tabelle 3 sind die „netto“ Verzögerungen exemplarisch für die Paketgröße 170 Bytes aufgeführt. Netto bedeutet, dass die in Kapitel 8.1 gemessene Verzögerung ohne geladene Link-Emulationssoftware abgezogen wurde. Man kann gut erkennen, dass die Mittelwerte bei Dummynet für kurze Delays etwas zu hoch ausfallen und die Mittelwerte bei hohen Delays etwas zu gering ausfallen. TC/NetEm trifft die gewünschte Verzögerung stets sehr präzise.

Das Dummynet ungenauer ist als NetEm lässt sich dadurch erklären, dass Dummynet auf die normale standardisierte Uhr zurückgreift, deren Auflösung von dem „HZ“ Kernel-Parameter²⁸ abhängt, während NetEm auf neuere und linuxspezifischen Präzisionsuhren („high resolution timers“)[22] zurückgreifen kann.

8.6 Modellierung von Verzögerungsschwankungen

Da bei Dummynet Verzögerungsschwankungen/Jitter nur indirekt über verschiedene Pipes modelliert werden können, ist, wie in Abbildung 12 zu sehen²⁹, nur eine relativ grobe Annäherung an eine statistische Verteilung möglich. Diese ist mit entsprechenden Auf-

²⁸Gibt die Zahl der Zählerschritte pro Sekunde an. Im verwendeten Testsystem mit einem Wert von 250 Hertz.

²⁹Exemplarisch am Beispiel von 1500 Byte großen Paketen. Andere Paketgrößen weisen ein vergleichbares Verhalten auf.

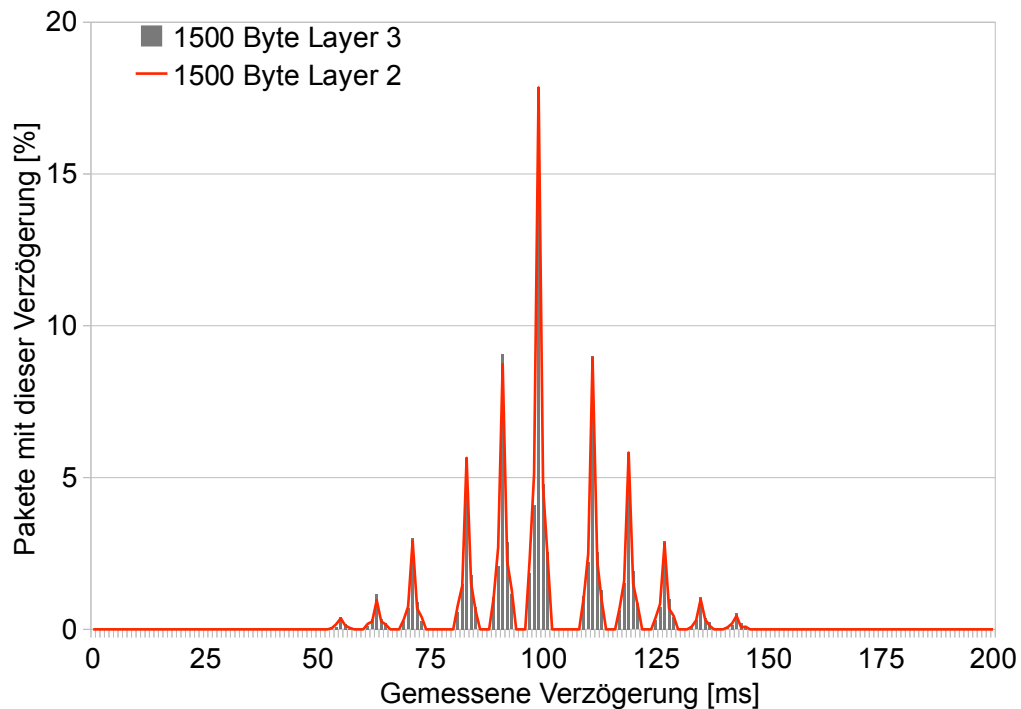


Abbildung 12: Modellierung von Jitter mit Dummysnet. Ziel 100ms Delay mit einer Normalverteilung $\pm 50\text{ms}$

wand, deutliche Erhöhung der Anzahl von verwendeten Pipes, verbesserungsfähig, wird aber vollständig dem Experimentator überlassen.

Bei NetEm hingegen werden wie in Abbildung 13 zu erkennen³⁰, nur etwa 70% der Pakete im konfigurierten Bereich von 50ms bis 150ms verzögert. Jeweils ca. 14% weisen noch höhere bzw. niedrigere Verzögerungen auf.

Sowohl bei Dummysnet als auch bei NetEm gab es keine signifikanten Unterschiede zwischen der Messung auf Layer 2 und Layer 3.

³⁰Exemplarisch am Beispiel von 1500 Byte großen Paketen. Andere Paketgrößen weisen ein vergleichbares Verhalten auf.

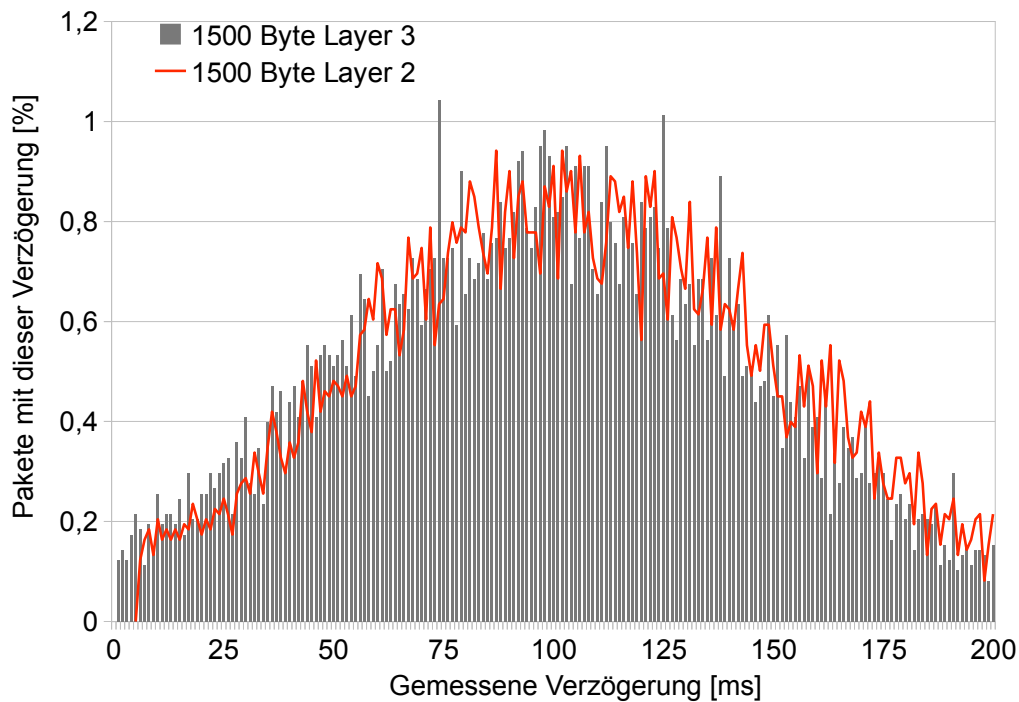


Abbildung 13: Modellierung von Jitter mit NetEm. Ziel 100ms Delay mit einer Normalverteilung ± 50 ms

Teil V

Zusammenfassung

9 Zusammenfassung & Ausblick

9.1 Zusammenfassung

In dieser Arbeit wurde die Linuxportierung von Dummynet um die Filterung auf Layer 2 erweitert und mit Messreihen mit der existierenden Implementierung von TC/NetEm verglichen. Beide Link-Emulatoren wurden zudem auf Unterschiede bei der Verwendung auf Layer 2 und Layer 3 geprüft.

Wie im Vorfeld erwartet wurde, zeigten sich keine signifikanten Unterschiede zwischen der Filterung auf Layer 2 und Layer 3. Beide Link-Emulatoren (Dummynet und TC/NetEm) sind grundsätzlich geeignet, um in virtuellen Netzen – und natürlich auch in Hardware existierenden Netzen – Verbindungsparameter zu emulieren.

Gegen DummyNet spricht, dass DummyNet bei der Bandbreitenlimitierung eine sehr hohe Paketverlustrate hat und allgemein einen sehr hohen Ressourcenbedarf besitzt. Für DummyNet spricht, dass es auf mehreren Plattformen verfügbar ist.

Für NetEm spricht, dass es sehr sparsam mit den Ressourcen umgeht. NetEm ist allerdings nur für Linux verfügbar und entspricht bei der Modellierung von Verzögerungsschwankungen nicht dem erwarteten Verhalten.

Beide Emulatoren beachten bei Bandbreitenbegrenzung nicht das Bandbreitenlatenzprodukt, so dass bei Untersuchungen, wo dies Auswirkungen haben kann, dies manuell approximiert werden muss.

9.2 Ausblick

Die in dieser Arbeit begonnenen Arbeiten bieten vielfältige Anknüpfungspunkte für weitere Arbeiten. Einige Bereiche werden im Folgenden angerissen.

Im Bereich der Portierung von DummyNet auf Linux sind dies insbesondere die Einbindung der Hooks auf der Bridging-Ebene, so dass auch für Pakete, die über eine Bridge laufen Filter möglich sind, was ein sinnvoller Ansatzpunkt für eine weitere Arbeit sein könnte. Durch geschickte Wahl der Hooks sollte auch eine Filterung nach Ziel-MAC-Adresse bei ausgehenden Paketen damit ermöglicht werden. Der im Vergleich zu TC/NetEm signifikant höhere Ressourcenbedarf und die konstante Verzögerung bei aktivierter Bandbreitenbegrenzung sind auch Bereiche, die näher untersucht werden könnten.

Für NetEm wäre noch zu untersuchen, wo die Ursache für die Abweichung bei der Emulation von Verzögerungsschwankungen zu suchen ist. Zusätzlich wäre für TC und damit auch für NetEm, eine Anpassung zur komfortablen Filterung auf MAC-Adressen sinnvoll.

Teil VI

Anhang

A Skripte für die Messungen

A.1 Datenrate

TC/NetEm

```
1 #!/bin/bash

3 set -o nounset

5 #
  # Messung für eine Konfigurierte Bandbreiten +- 10%
7 # zum Zeigen wie gut die von der jeweiligen Technik erreicht wird.

9
  # Die Zielbandbreite für die Läufe in KBits/s
11 bandwidths="66 131 262 524 1049 2097 4194 8389 16777"

13 # Payload der udp-Pakete in Bytes
  payloads="128 256 512 1024 1448"
15
  # Dauer eines Experiments
17 experimentdauer=30

19 # Von Hand vorzubereitende Konfiguration vor dem Skriptlauf
  #
21 # auf delta nuttcp-Server starten
  # nuttcp -S --nofork
23 #
  # auf beta das basissetup Konfigurieren,
25 # genaues Setup wird automatisch eingestellt
  #
27 #
  # TC/NetEm Layer 3:
29 # tc qdisc add dev vlan10 root handle 1:0 netem
  # tc qdisc add dev vlan10 parent 1:1 handle 10: |
31 #   tbf rate 10kbit buffer 2"kb" latency 20ms
  # modprobe ifb
```

```

33 # ip link set dev ifb0 up
    # tc qdisc add dev vlan10 ingress
35 # tc filter add dev vlan10 parent ffff: protocol ip u32 |
    #   match ip src 172.23.1.0/24 flowid 1:1 |
37 #   action mirrored egress redirect dev ifb0
    # tc qdisc add dev ifb0 root handle 1:0 netem
39 # tc qdisc add dev ifb0 parent 1:1 handle 10: |
    #   tbf rate 10kbit buffer 2kb" latency 20ms
41 #
    #
43 # TC/NetEm Layer 2:
    # tc qdisc add dev vlan10 root handle 1:0 netem
45 # tc qdisc add dev vlan10 parent 1:1 handle 10: |
    #   tbf rate 10kbit buffer 2"kb" latency 20ms
47 # modprobe ifb
    # ip link set dev ifb0 up
49 # tc qdisc add dev vlan10 ingress
    # tc filter add dev vlan10 parent ffff: protocol ip u32 |
51 #   match u16 0x4ddb 0xFFFF at -4 match u32 0x000102df 0xFFFFFFFF at -8 |
    #   flowid 1:1 action mirrored egress redirect dev ifb0
53 # tc qdisc add dev ifb0 root handle 1:0 netem
    # tc qdisc add dev ifb0 parent 1:1 handle 10: |
55 #   tbf rate 10kbit buffer 2kb" latency 20ms

57 # langsam an die Bandbreite herantasten, dabei jeweils den Paketverlust
    # messen (sollte 0 sein bei <= bandbreite)
59 #
    # Dabei die 20 Byte IP-Header-Overhead und 8 Byte vom UDP-Header
61 # berücksichtigen, die nicht von nuttcp berücksichtigt werden
    # Bei tc zusätzlich 14 byte Ethernet-Header + 4 Byte CRC
63
    for bandwidth in $bandwidths; do
65         echo
        echo
67         echo "$bandwidth kBit/s"
        echo "-----"
69         echo
        # Bandbreite auf Beta setzen
71         # Puffer-Slots Berechnen, grob vergleichbar mit dummysnet (kb vs. slots
            )
            puffer=$(( $bandwidth / 500 )
73         if [ $puffer -lt 2 ]; then
            puffer=2
75         elif [ $puffer -gt 100 ]; then

```

```

    puffer=100
77  fi
    puffer=$puffer"kb"
79  # nun die Pipes anpassen
    NetEmbw=$bandwidth"kbit"
81  ssh -l root 132.252.152.152 "tc qdisc change dev vlan1 parent 1:1 \
    handle 10: tbf rate $NetEmbw buffer $puffer latency 20ms"
83  ssh -l root 132.252.152.152 "tc qdisc change dev ifb0 parent 1:1 \
    handle 10: tbf rate $NetEmbw buffer $puffer latency 20ms"
85
    # nun für alle der Nutzdatengrößen
87  for payload in $payloads; do
    # ersteinmal sicherstellen das ARP-Caches & Co gefüllt sind
89  ssh -l root 132.252.152.150 "ping -n -c 3 172.23.2.3 > /dev/null"

91  # Ping Messen fuer Bandbreitenlatenzprodukt
    echo -e "$payload: \t"$(ssh -l root 132.252.152.150 \
93  "ping -n -q -c 10 172.23.2.3" | tail -2) | \
    tee -a Ergebnisse/Bandbreite/netEm-$bandwidth"-ping.txt"
95
    # Durchsatz per TCP
97  ( echo -e -n "$payload:\t"
    ssh -l root 132.252.152.150 "nuttcp -T$experimentdauer \
99  -l $payload 172.23.2.3" ) \
    | tee -a Ergebnisse/Bandbreite/netEm-$bandwidth"-tcp.txt"
101
    # UDP-Tests
103  for (( prozent=90; $prozent <= 110; prozent++ ))
    do
105  ( nuttcprate=$(echo "scale=2; ($bandwidth * $prozent *
    $payload) / (100 * ($payload +28+18) )" | bc -l)"k"
    echo -e -n "$payload:\t"
107  ssh -l root 132.252.152.150 "nuttcp -u -R$nuttcprate \
    -T$experimentdauer -l $payload 172.23.2.3" ) \
109  | tee -a Ergebnisse/Bandbreite/netEm-$bandwidth"-udp-"
    $prozent".txt"

    done
111  done
done

```

Dummynet

```
#!/bin/bash
```

2

```

    set -o nounset
4
#
6 # Messung für eine Konfigurierte Bandbreiten +- 10%
  # zum Zeigen wie gut die von der jeweiligen Technik erreicht wird.
8
10 # Die Zielbandbreite für die Läufe in KBits/s
    bandwidths="66 131 262 524 1049 2097 4194 8389 16777"
12
    # Payload der udp-Pakete in Bytes
14 payloads="128 256 512 1024 1448"
16 # Dauer eines Experiments
    experimentdauer=30
18
    # Von Hand vorzubereitende Konfiguration vor dem Skriptlauf
20 #
    # auf delta nuttcp-Server starten
22 # nuttcp -S --nofork
    #
24 # auf beta das basissetup Konfigurieren,
    # genaues Setup wird automatisch eingestellt
26 #
    #
28 # Dummynet Layer 3:
    # insmod /usr/local/src/ipfw_linux-20090724/dummynet/ipfw_mod.ko
30 # ipfw pipe 4 config bw 10Kbit/s queue 1
    # ipfw add pipe 4 src-ip 172.23.1.0/24 out
32 # ipfw pipe 5 config bw 10Kbit/s queue 1
    # ipfw add pipe 5 dst-ip 172.23.1.0/24 out
34 #
    #
36 # Dummynet Layer 2:
    # auf beta das basissetup Konfigurieren,
38 # genaues Setup wird automatisch eingestellt
    # insmod /usr/local/src/Development/ipfw_linux-20090724/dummynet/ipfw_mod.
        ko
40 # ipfw pipe 4 config bw 10Kbit/s queue 1
    # ipfw add pipe 4 src-ether 00:01:02:df:4d:db in
42 # ipfw pipe 5 config bw 10Kbit/s queue 1
    # ipfw add pipe 5 dst-ether 00:01:02:df:4d:db out
44 #
    # langsam an die Bandbreite herantasten, dabei jeweils den Paketverlust

```

```

46 # messen (sollte 0 sein bei <= bandbreite)
   #
48 # Dabei die 20 Byte IP-Header-Overhead und 8 Byte vom UDP-Header
   # berücksichtigen, die nicht von nuttcp berücksichtigt werden
50
   for bandwidth in $bandwidths; do
52     echo
   echo
54     echo "$bandwidth kBit/s"
   echo "-----"
56     echo
   # Bandbreite auf Beta setzen
58     # Puffer-Slots Berechnen, nicht zu klein, und nicht >100
   # (ist ein Limit von Dummynet)
60     puffer=$(( $bandwidth / 250 ))
   if [ $puffer -lt 2 ]; then
62         puffer=2
   elif [ $puffer -gt 100 ]; then
64         puffer=100
   fi
66     # nun die Pipes anpassen
   dummynetbw=$(( $bandwidthKbit/s ))
68     ssh -l root 132.252.152.152 "ipfw pipe 4 config bw $dummynetbw \
   queue $puffer"
70     ssh -l root 132.252.152.152 "ipfw pipe 5 config bw $dummynetbw \
   queue $puffer"
72
   # nun für alle der Nutzdatengrößen
74     for payload in $payloads; do
   # ersteinmal sicherstellen das ARP-Caches & Co gefüllt sind
76     ssh -l root 132.252.152.150 "ping -n -c 3 172.23.2.3 > /dev/null"

78     # Ping Messen fuer Bandbreitenlatenzprodukt
   echo -e "$payload: \t"$(ssh -l root 132.252.152.150 \
80     "ping -n -q -c 10 172.23.2.3" | tail -2) | \
   tee -a Ergebnisse/Bandbreite/netEm-$bandwidth"-ping.txt"
82
   # Durchsatz per TCP
84     ( echo -e -n "$payload:\t"
   ssh -l root 132.252.152.150 "nuttcp -T$experimentdauer \
86     -l $payload 172.23.2.3" ) \
   | tee -a Ergebnisse/Bandbreite/netEm-$bandwidth"-tcp.txt"
88
   # UDP-Tests

```

```

90     for (( prozent=90; $prozent <= 110; prozent++ ))
91     do
92         (
93             nuttcprate=$(echo "scale=2; ($bandwidth * $prozent *
94             $payload) / (100 * ($payload +28) )" | bc -l)"k"
95             echo -e -n "$payload:\t"
96             ssh -l root 132.252.152.150 "nuttcp -u -R$nuttcprate \
97             -T$experimentdauer -l $payload 172.23.2.3" ) \
98             | tee -a Ergebnisse/Bandbreite/netEm-$bandwidth"-udp-"
99             $prozent".txt"
100        done
101    done
102 done

```

A.2 Verzögerung

```

1  #!/bin/bash
2
3  set -o nounset
4
5  #
6  # Messung für Verzögerungen/Delay
7  #
8
9  # Verzögerungen
10 delays="5 10 20 50 100 200 500 1000"
11
12 # Payload der ICMP-Pakete in Bytes
13 payloads="128 256 512 1024 1448"
14
15 # Dauer der Testausführung
16 experimentdauer=10000
17
18 # auf beta das Basissetup Konfigurieren,
19 # genaues Setup wird automatisch eingestellt
20 #
21 # DummyNet Layer 3:
22 # insmod /usr/local/src/ipfw_linux-20090724/dummyNet/ipfw_mod.ko
23 # ipfw pipe 4 config delay 0ms
24 # ipfw add pipe 4 src-ip 172.23.1.0/24 out
25 #
26 # DummyNet Layer 2:
27 # insmod /usr/local/src/Development/ipfw_linux-20090724/dummyNet/ipfw_mod.ko

```

```

# ipfw pipe 4 config delay 0ms
29 # ipfw add pipe 4 src-ether 00:01:02:df:4d:db in
#
31 #
# TC/NetEm Layer 3:
33 # tc qdisc add dev vlan10 root handle 1:0 netem
# modprobe ifb
35 # ip link set dev ifb0 up
# tc qdisc add dev vlan10 ingress
37 # tc filter add dev vlan10 parent ffff: protocol ip u32 |
# match ip src 172.23.1.0/24 flowid 1:1 |
39 # action mirrored egress redirect dev ifb0
# tc qdisc add dev ifb0 root handle 1:0 netem
41 #
# TC/NetEm Layer 2:
43 # tc qdisc add dev vlan10 root handle 1:0 netem
# modprobe ifb
45 # ip link set dev ifb0 up
# tc qdisc add dev vlan10 ingress
47 # tc filter add dev vlan10 parent ffff: protocol ip u32 |
# match u16 0x4ddb 0xFFFF at -4 match u32 0x000102df 0xFFFFFFFF at -8 |
49 # flowid 1:1 action mirrored egress redirect dev ifb0
# tc qdisc add dev ifb0 root handle 1:0 netem
51
# Prefix für den Dateinamen
53 filenamePrefix=netEm

55 for delay in $delays; do
    echo
57    echo
    echo "$delay ms"
59    echo "_____ "
    echo
61    delayParam=$delay"ms"
    # NetEm
63    ssh -l root 132.252.152.152 "tc qdisc change dev ifb0 root handle 1:0
    \
        netem delay $delayParam"
65    # Dummynet
    # ssh -l root 132.252.152.152 "ipfw pipe 4 config delay $delayParam"
67
    # nun für alle der Nutzdatengrößen
69    for payload in $payloads; do
        # ersteinmal sicherstellen das ARP-Caches & Co gefüllt sind

```

```

71     ssh -l root 132.252.152.150 "ping -n -c 3 172.23.2.3 > /dev/null"
73     # Delay-tests
       echo -e "$payload:\t" $(ssh -l root 132.252.152.150 "ping -q \
75         -c $experimentdauer -i 0.01 -s $payload 172.23.2.3" ) \
         | tee -a Ergebnisse/delay/$filenamePrefix-$delay".txt"
77     done
done

```

A.3 Paketverluste

A.3.1 Baseline

Ein Beispiel für die Messung der Paketverlustrate bei der Baseline-Messung. Für die anderen Messungen war nur der Konfigurationsabschnitt modifiziert.

```

#!/bin/bash
2
set -o nounset
4
#
6 # Einfache Messung welche Bandbreite mit welchem Paketverlust erreicht
   wird.
   # 500 Kbit/s Spruenge
8 #
10 # Payload der udp-Pakete in Bytes
    payloads="100 750 1448"
12
    # Dauer jeder Testausführung in Sekunden
14 experimentdauer=30
16 # Welches System
    prefix="L2dumynet"
18
    # rebooten, damit sicher nix mehr geladen ist
20 ssh -l root 132.252.152.152 "reboot"
    sleep 300
22
    # Konfiguration setzen
24 ssh -l root 132.252.152.152 "insmod /usr/local/src/Development/ipfw_linux
    -20090724/dumynet/ipfw_mod.ko"
    ssh -l root 132.252.152.152 "ipfw pipe 4 config bw 100Mbit/s queue 100"
26 ssh -l root 132.252.152.152 "ipfw add pipe 4 \

```

```

    src-ether 00:01:02:df:4d:db in"
28 ssh -l root 132.252.152.152 "ipfw pipe 5 config bw 100Mbit/s queue 100"
    ssh -l root 132.252.152.152 "ipfw add pipe 5 \
30     dst-ether 00:01:02:df:4d:db out"

32 # Daten sammeln

34 for payload in $payloads; do
    echo
36     echo
    echo "_____ "
38     echo "Sende UDP-Pakete mit $payload Byte Nutzdaten, d.h. auf \
        Ethernet-Ebene $[ $payload + 28 +14 +4 ] Byte"
40     echo "_____ "
    echo
42     for (( schritt=1; $schritt <= 200; schritt++ ))
        do
44         # Bei der Bandbreitenberechnung alle Header berücksichtigen
            # (UDP/IP, Ethernet + Ethernet-Checksume)
46         bandbreite=$(echo "scale=2; \
                ($schritt * 500 * $payload) / ($payload +28+14+4)" | bc -l)k
48         echo -e -n $(echo "scale=1; $schritt * 0.5" | bc -l)" MBit/s:\t" \
            | tee -a Ergebnisse/$prefix-"$payload".txt"
50         ssh -l root 132.252.152.150 "nuttcp -u -R$bandbreite \
            -T$experimentdauer -l $payload 172.23.2.3 " \
52         | tee -a Ergebnisse/$prefix-"$payload".txt"
            sleep 2
54         done
        done
done

```

Zur Auswertung der Logfiles wurde das folgende Skript genutzt:

```

1 #!/bin/bash

3 set -o nounset
    basis=$1
5 bw=$2
    cat $basis-$bw.txt | cut -d "/" -f 5 | cut -d " " -f 2 | sed s/\\.\\.\\.//g |
    awk '{CONVFMT="%.6g"; print $1/100}'

```

A.3.2 Vorgegebene Verlustrate

```
#!/bin/bash
```

```
2
```

```

    set -o nounset
4
    #
6    # Paketverluste
    losses="0.001 0.005 0.01 0.05 0.10"
8
    # Payload der Pakete in Bytes
10    payload="100"
    # Dauer der Testausführung
12    experimentdauer=30
    # Payload der udp-Pakete in Bytes
14    payloads="100 750 1448"

16
    # Welches System
18    prefix="L3tcNetEm"

20    # rebooten, damit sicher nix mehr geladen ist
    ssh -l root 132.252.152.152 "reboot"
22    sleep 300

24    # Konfiguration setzen
    ssh -l root 132.252.152.152 "tc qdisc add dev vlan10 root handle 1:0 netem
    "
26    ssh -l root 132.252.152.152 "modprobe ifb"
    ssh -l root 132.252.152.152 "ip link set dev ifb0 up"
28    ssh -l root 132.252.152.152 "tc qdisc add dev vlan10 ingress"
    ssh -l root 132.252.152.152 "tc filter add dev vlan10 parent ffff: \
30    protocol ip u32 match ip src 172.23.1.0/24 flowid 1:1 \
        action mirred egress redirect dev ifb0"
32    ssh -l root 132.252.152.152 "tc qdisc add dev ifb0 root handle 1:0 netem"

34    # Daten sammeln

36    for loss in $losses; do
        echo
38        echo
        echo "$loss %"
40        echo "-----"
        echo
42        NetEmbW=0$( echo $loss*100 | bc )"%"
        ssh -l root 132.252.152.152 "tc qdisc change dev ifb0 root handle 1:0
        \
44        netem loss $NetEmbW"

```

```

46  # ersteinmal sicherstellen das ARP-Caches & Co gefüllt sind
    ssh -l root 132.252.152.150 "ping -n -c 3 172.23.2.3 > /dev/null"
48  ssh -l root 132.252.152.150 "nuttcp -u -R684.93k -T$experimentdauer \
    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-01MBit.log
50  ssh -l root 132.252.152.150 "nuttcp -u -R13698.63k -T$experimentdauer
    \
    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-20MBit.log
52  ssh -l root 132.252.152.150 "nuttcp -u -R27397.26k -T$experimentdauer
    \
    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-40MBit.log
54  ssh -l root 132.252.152.150 "nuttcp -u -R51369.86k -T$experimentdauer
    \
    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-75MBit.log
56  done

58  # Welches System
    prefix="L2tcNetEm"
60
    # rebooten, damit sicher nix mehr geladen ist
62  ssh -l root 132.252.152.152 "reboot"
    sleep 300
64
    # Konfiguration setzen
66  ssh -l root 132.252.152.152 "tc qdisc add dev vlan10 root handle 1:0 netem
    "
    ssh -l root 132.252.152.152 "modprobe ifb"
68  ssh -l root 132.252.152.152 "ip link set dev ifb0 up"
    ssh -l root 132.252.152.152 "tc qdisc add dev vlan10 ingress"
70  ssh -l root 132.252.152.152 "tc filter add dev vlan10 parent ffff: \
    protocol ip u32 match u16 0x4ddb 0xFFFF at -4 \
72  match u32 0x000102df 0xFFFFFFFF at -8 flowid 1:1 \
    action mirrored egress redirect dev ifb0"
74  ssh -l root 132.252.152.152 "tc qdisc add dev ifb0 root handle 1:0 netem"

76  # Daten sammeln

78  for loos in $looses; do
    echo
80  echo
    echo "$loos %"
82  echo "_____ "
    echo
84  NetEmbW=0$( echo $loos*100 | bc )%"

```

```

ssh -l root 132.252.152.152 "tc qdisc change dev ifb0 root handle 1:0
\
86     netem loss $NetEmbW"

88     # ersteinmal sicherstellen das ARP-Caches & Co gefüllt sind
ssh -l root 132.252.152.150 "ping -n -c 3 172.23.2.3 > /dev/null"
90     ssh -l root 132.252.152.150 "nuttcp -u -R684.93k -T$experimentdauer \
    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-01MBit.log
92     ssh -l root 132.252.152.150 "nuttcp -u -R13698.63k -T$experimentdauer
    \
    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-20MBit.log
94     ssh -l root 132.252.152.150 "nuttcp -u -R27397.26k -T$experimentdauer
    \
    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-40MBit.log
96     ssh -l root 132.252.152.150 "nuttcp -u -R51369.86k -T$experimentdauer
    \
    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-75MBit.log
98 done

100
    # Welches System
102     prefix="L3dummysnet"

104     # rebooten, damit sicher nix mehr geladen ist
ssh -l root 132.252.152.152 "reboot"
106     sleep 300

108     # Konfiguration setzen
ssh -l root 132.252.152.152 "insmod /usr/local/src/ipfw_linux-20090724/
    dummysnet/ipfw_mod.ko"
110     ssh -l root 132.252.152.152 "/usr/local/src/ipfw_linux-20090724/ipfw/ipfw
    pipe 4 config"
ssh -l root 132.252.152.152 "/usr/local/src/ipfw_linux-20090724/ipfw/ipfw
    add pipe 4 src-ip 172.23.1.0/24 out"
112

    # Daten sammeln
114
    for loos in $looses; do
116         echo
116         echo
118         echo "$loos %"
118         echo "_____ "
120         echo
dummysnetbw=$loos

```

```

122 ssh -l root 132.252.152.152 "ipfw pipe 4 config plr $dummynetbw"

124 # ersteinmal sicherstellen das ARP-Caches & Co gefüllt sind
ssh -l root 132.252.152.150 "ping -n -c 3 172.23.2.3 > /dev/null"
126 ssh -l root 132.252.152.150 "nuttcp -u -R684.93k -T$experimentdauer \
-l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-01MBit.log
128 ssh -l root 132.252.152.150 "nuttcp -u -R13698.63k -T$experimentdauer
\
-l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-20MBit.log
130 ssh -l root 132.252.152.150 "nuttcp -u -R27397.26k -T$experimentdauer
\
-l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-40MBit.log
132 ssh -l root 132.252.152.150 "nuttcp -u -R51369.86k -T$experimentdauer
\
-l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-75MBit.log

134
done
136
# Welches System
138 prefix="L2dummynet"

140 # rebooten, damit sicher nix mehr geladen ist
ssh -l root 132.252.152.152 "reboot"
142 sleep 300

144 # Konfiguration setzen
ssh -l root 132.252.152.152 "insmod /usr/local/src/Development/ipfw_linux
-20090724/dummynet/ipfw_mod.ko"
146 ssh -l root 132.252.152.152 "/usr/local/src/Development/ipfw_linux
-20090724/ipfw/ipfw pipe 4 config"
ssh -l root 132.252.152.152 "/usr/local/src/Development/ipfw_linux
-20090724/ipfw/ipfw add pipe 4 src-ether 00:01:02:df:4d:db in"

148
# Daten sammeln
150
for loos in $looses; do
152 echo
153 echo
154 echo "$loos %"
echo "_____ "
156 echo
dummynetbw=$loos
158 ssh -l root 132.252.152.152 "ipfw pipe 4 config plr $dummynetbw"

```

```

160  # ersteinmal sicherstellen das ARP-Caches & Co gefüllt sind
      ssh -l root 132.252.152.150 "ping -n -c 3 172.23.2.3 > /dev/null"
162  ssh -l root 132.252.152.150 "nuttcp -u -R684.93k -T$experimentdauer \
      -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-01MBit.log
164  ssh -l root 132.252.152.150 "nuttcp -u -R13698.63k -T$experimentdauer
      \
      -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-20MBit.log
166  ssh -l root 132.252.152.150 "nuttcp -u -R27397.26k -T$experimentdauer
      \
      -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-40MBit.log
168  ssh -l root 132.252.152.150 "nuttcp -u -R51369.86k -T$experimentdauer
      \
      -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-75MBit.log
170  done

172  # Welches System
      prefix="L3dummynetBW"
174
      # rebooten, damit sicher nix mehr geladen ist
176  ssh -l root 132.252.152.152 "reboot"
      sleep 300
178
      # Konfiguration setzen
180  ssh -l root 132.252.152.152 "insmod /usr/local/src/ipfw_linux-20090724/
      dummynet/ipfw_mod.ko"
      ssh -l root 132.252.152.152 "/usr/local/src/ipfw_linux-20090724/ipfw/ipfw
      pipe 4 config"
182  ssh -l root 132.252.152.152 "/usr/local/src/ipfw_linux-20090724/ipfw/ipfw
      add pipe 4 src-ip 172.23.1.0/24 out"

184  # Daten sammeln

186  for loos in $looses; do
      echo
188  echo
      echo "$loos %"
190  echo "-----"
      echo
192  dummynetbw=$loos
      ssh -l root 132.252.152.152 "ipfw pipe 4 config plr $dummynetbw bw 100
      Mbit/s queue 100"
194
      # ersteinmal sicherstellen das ARP-Caches & Co gefüllt sind
196  ssh -l root 132.252.152.150 "ping -n -c 3 172.23.2.3 > /dev/null"

```

```

198     ssh -l root 132.252.152.150 "nuttcp -u -R684.93k -T$experimentdauer \  

    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-01MBit.log
200     ssh -l root 132.252.152.150 "nuttcp -u -R13698.63k -T$experimentdauer \  

    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-20MBit.log
202     ssh -l root 132.252.152.150 "nuttcp -u -R27397.26k -T$experimentdauer \  

    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-40MBit.log
204     ssh -l root 132.252.152.150 "nuttcp -u -R51369.86k -T$experimentdauer \  

    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-75MBit.log
done
206
208 # Welches System
prefix="L2dummysnetBW"

210 # rebooten, damit sicher nix mehr geladen ist
ssh -l root 132.252.152.152 "reboot"
212 sleep 300

214 # Konfiguration setzen
ssh -l root 132.252.152.152 "insmod /usr/local/src/Development/ipfw_linux
-20090724/dummysnet/ipfw_mod.ko"
216 ssh -l root 132.252.152.152 "/usr/local/src/Development/ipfw_linux
-20090724/ipfw/ipfw pipe 4 config"
ssh -l root 132.252.152.152 "/usr/local/src/Development/ipfw_linux
-20090724/ipfw/ipfw add pipe 4 src-ether 00:01:02:df:4d:db in"
218
220 # Daten sammeln
for loos in $looses; do
222     echo
224     echo "$loos %"
226     echo "-----"
228     dummysnetbw=$loos
ssh -l root 132.252.152.152 "ipfw pipe 4 config plr $dummysnetbw bw 100
Mbit/s queue 100"

230 # ersteinmal sicherstellen das ARP-Caches & Co gefüllt sind
ssh -l root 132.252.152.150 "ping -n -c 3 172.23.2.3 > /dev/null"
232     ssh -l root 132.252.152.150 "nuttcp -u -R684.93k -T$experimentdauer \  

    -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-01MBit.log

```

```

234 ssh -l root 132.252.152.150 "nuttcp -u -R13698.63k -T$experimentdauer
    \
      -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-20MBit.log
236 ssh -l root 132.252.152.150 "nuttcp -u -R27397.26k -T$experimentdauer
    \
      -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-40MBit.log
238 ssh -l root 132.252.152.150 "nuttcp -u -R51369.86k -T$experimentdauer
    \
      -l $payload 172.23.2.3" | tee -a Ergebnisse/$prefix-75MBit.log
240 done

```

A.4 Verzögerunsschwankungen

Konfiguration Die Konfiguration ist für DummyNet und NetEm nur exemplarisch für Layer 2 abgedruckt, Layer 3 wurde äquivalent Konfiguriert (nur IP statt MAC als Filterkriterium).

```

# DummyNet
2 ipfw pipe 6 config delay 100ms ipfw add prob 0.3108 pipe 6 src-ether
  00:01:02:df:4d:db in
  ipfw pipe 7 config delay 111ms ipfw add prob 0.2329 pipe 7 src-ether
    00:01:02:df:4d:db in ipfw pipe 8 config delay 89ms ipfw add prob 0.3036
      pipe 8 src-ether 00:01:02:df:4d:db in
4 ipfw pipe 9 config delay 119ms ipfw add prob 0.2806 pipe 9 src-ether
  00:01:02:df:4d:db in ipfw pipe 10 config delay 81ms ipfw add prob
    0.3900 pipe 10 src-ether 00:01:02:df:4d:db in
  ipfw pipe 11 config delay 128ms ipfw add prob 0.3224 pipe 11 src-ether
    00:01:02:df:4d:db in ipfw pipe 12 config delay 72ms ipfw add prob
      0.4758 pipe 12 src-ether 00:01:02:df:4d:db in
6 ipfw pipe 13 config delay 136ms ipfw add prob 0.3571 pipe 13 src-ether
  00:01:02:df:4d:db in ipfw pipe 14 config delay 64ms ipfw add prob
    0.5556 pipe 14 src-ether 00:01:02:df:4d:db in
  ipfw pipe 15 config delay 144ms ipfw add prob 0.5 pipe 15 src-ether
    00:01:02:df:4d:db in ipfw pipe 16 config delay 56ms ipfw add prob 1
      pipe 16 src-ether 00:01:02:df:4d:db in
8
# NetEm
10 tc qdisc add dev vlan1 root handle 1:0 netem
  modprobe ifb
12 ip link set dev ifb0 up
  tc qdisc add dev vlan1 ingress
14 tc filter add dev vlan1 parent ffff: protocol ip u32 match u16 0x4ddb 0
  xFFFF at -4 match u32 0x000102df 0xFFFFFFFF at -8 flowid 1:1 action
  mirrored egress redirect dev ifb0

```

```
tc qdisc add dev ifb0 root handle 1:0 netem delay 100ms 50ms distribution
normal
```

Messung Sammlung der Daten, für NetEm nur geänderte Dateinamen:

```
1 #!/bin/bash

3 set -o nounset

5 # Payload der Pakete in Bytes
  payloads="128 256 512 1024 1448"
7
  # Dauer der Testausführung
9 experimentdauer=10000

11 for payload in $payloads; do
    # ersteinmal sicherstellen das ARP-Caches & Co gefüllt sind
13 ssh -l root 132.252.152.150 "ping -n -c 3 172.23.2.3 > /dev/null"
    # Delay-tests
15 echo -e "$payload:"
    ssh -l root 132.252.152.150 "ping -c$experimentdauer -i 0.01 -s
      $payload 172.23.2.3" | tee -a Ergebnisse/jitter/dummysnet-$payload"
      .txt"
17 done
```

Auswertung der Daten: Für die Auswertung wird awk genutzt.

```
1 #!/bin/bash
  file=$1
3 bw=$2
  fgrep from $file-$bw.txt | cut -d "=" -f 4 | cut -d " " -f 1 | awk '{
5   CONVEMT = "%.20f"
   traffic[$1+0] += 1
7   }
  END {
9   i=0
   while (i <= 200) {
11     print int(traffic[i])
     i++
13   }
   }'
```

A.5 CPU-Last

Messungen Die Messungen wurden mit dem folgenden Skript durchgeführt, die Messungen für Dummynet mit größeren Paketen äquivalent, mit geänderter Nutzlast.

```
1 #!/bin/bash

3 set -o nounset

5 # globale Variable
  payload=100
7 experimentdauer=150
  messdauer=130
9 wartezeit=30

11 # Baseline, ohne jegliches Modul
  prefix=baseline
13
  # rebooten, damit sicher nix mehr geladen ist
15 ssh -l root 132.252.152.152 "reboot"
  sleep 300
17
  # nun für jedes Mbit Messung durchführen
19 for schritt in {1..75}; do
    echo -e -n $schritt" MBit/s:\t" >>cpu/$prefix-nuttcp.log
21   bandbreite=$(echo "scale=2; ($schritt * 1024 * $payload) / ($payload
    +28+14)" | bc -l)k
    ssh -l root 132.252.152.150 "nuttcp -u -R$bandbreite -
      T$experimentdauer -l $payload 172.23.2.3" >>cpu/$prefix-nuttcp.log
    &
23   sleep 5
    ssh -l root 132.252.152.152 "vmstat -n 1 $messdauer" > cpu/$prefix-
      $schritt".log"
25   sleep $wartezeit
  done
27

29
  # TC/NetEm - L3
31 prefix=L3netem

33 # rebooten, damit sicher nix mehr geladen ist
  ssh -l root 132.252.152.152 "reboot"
35 sleep 300
```

```

37 # Konfiguration setzen
    ssh -l root 132.252.152.152 "tc qdisc add dev vlan1 root handle 1:0 netem"
39 ssh -l root 132.252.152.152 "tc qdisc add dev vlan1 parent 1:1 handle 10:
    tbf rate 100Mbit buffer 200kb latency 20ms"
    ssh -l root 132.252.152.152 "modprobe ifb"
41 ssh -l root 132.252.152.152 "ip link set dev ifb0 up"
    ssh -l root 132.252.152.152 "tc qdisc add dev vlan1 ingress"
43 ssh -l root 132.252.152.152 "tc filter add dev vlan1 parent ffff: protocol
    ip u32 match ip src 172.23.1.0/24 flowid 1:1 action mirred egress
    redirect dev ifb0"
    ssh -l root 132.252.152.152 "tc qdisc add dev ifb0 root handle 1:0 netem"
45 ssh -l root 132.252.152.152 "tc qdisc add dev ifb0 parent 1:1 handle 10:
    tbf rate 100Mbit buffer 200kb latency 20ms"

47 # nun für jedes Mbit Messung durchführen
    for schritt in {1..75}; do
49     echo -e -n "$schritt" MBit/s:\t" >>cpu/$prefix-nuttcp.log
        bandbreite=$(echo "scale=2; ($schritt * 1024 * $payload) / ($payload
            +28+14)" | bc -l)k
51     ssh -l root 132.252.152.150 "nuttcp -u -R$bandbreite -
        T$experimentdauer -l $payload 172.23.2.3" >>cpu/$prefix-nuttcp.log
        &
        sleep 5
53     ssh -l root 132.252.152.152 "vmstat -n 1 $messdauer" > cpu/$prefix-
        $schritt".log"
        sleep $wartezeit
55 done

57 # TC/NetEm - L2
    prefix=L2netem
59
    # rebooten, damit sicher nix mehr geladen ist
61 ssh -l root 132.252.152.152 "reboot"
    sleep 300
63
    # Konfiguration setzen
65 ssh -l root 132.252.152.152 "tc qdisc add dev vlan1 root handle 1:0 netem"
    ssh -l root 132.252.152.152 "tc qdisc add dev vlan1 parent 1:1 handle 10:
        tbf rate 100Mbit buffer 200kb latency 20ms"
67 ssh -l root 132.252.152.152 "modprobe ifb"
    ssh -l root 132.252.152.152 "ip link set dev ifb0 up"
69 ssh -l root 132.252.152.152 "tc qdisc add dev vlan1 ingress"

```

```

ssh -l root 132.252.152.152 "tc filter add dev vlan1 parent ffff: protocol
    ip u32 match u16 0x4ddb 0xFFFF at -4 match u32 0x000102df 0xFFFFFFFF
    at -8 flowid 1:1 action mirrored egress redirect dev ifb0"
71 ssh -l root 132.252.152.152 "tc qdisc add dev ifb0 root handle 1:0 netem"
ssh -l root 132.252.152.152 "tc qdisc add dev ifb0 parent 1:1 handle 10:
    tbf rate 100Mbit buffer 200kb latency 20ms"
73
# nun für jedes Mbit Messung durchführen
75 for schritt in {1..75}; do
    echo -e -n $schritt " MBit/s:\t" >>cpu/$prefix-nuttcp.log
77    bandbreite=$(echo "scale=2; ($schritt * 1024 * $payload) / ($payload
        +28+14)" | bc -l)k
    ssh -l root 132.252.152.150 "nuttcp -u -R$bandbreite -
        T$experimentdauer -l $payload 172.23.2.3" >>cpu/$prefix-nuttcp.log
        &
79    sleep 5
    ssh -l root 132.252.152.152 "vmstat -n 1 $messdauer" > cpu/$prefix-
        $schritt".log"
81    sleep $wartezeit
done
83
# Dummynet - L3
85 prefix=L3dummynet

87 # rebooten, damit sicher nix mehr geladen ist
ssh -l root 132.252.152.152 "reboot"
89 sleep 300

91 # Konfiguration setzen
# Konfiguration setzen
93 ssh -l root 132.252.152.152 "insmod /usr/local/src/ipfw_linux-20090724/
    dummynet/ipfw_mod.ko"
    ssh -l root 132.252.152.152 "/usr/local/src/ipfw_linux-20090724/ipfw/ipfw
    pipe 4 config bw 100Mbit/s queue 100"
95 ssh -l root 132.252.152.152 "/usr/local/src/ipfw_linux-20090724/ipfw/ipfw
    add pipe 4 src-ip 172.23.1.0/24 out"
    ssh -l root 132.252.152.152 "/usr/local/src/ipfw_linux-20090724/ipfw/ipfw
    pipe 5 config bw 100Mbit/s queue 100"
97 ssh -l root 132.252.152.152 "/usr/local/src/ipfw_linux-20090724/ipfw/ipfw
    add pipe 5 dst-ip 172.23.1.0/24 out"

99 # nun für jedes Mbit Messung durchführen
for schritt in {1..75}; do
101    echo -e -n $schritt " MBit/s:\t" >>cpu/$prefix-nuttcp.log

```

```

bandbreite=$(echo "scale=2; ($schritt * 1024 * $payload) / ($payload
+28+14)" | bc -l)k
103 ssh -l root 132.252.152.150 "nuttcp -u -R$bandbreite -
T$experimentdauer -l $payload 172.23.2.3" >>cpu/$prefix-nuttcp.log
&
sleep 5
105 ssh -l root 132.252.152.152 "vmstat -n 1 $messdauer" > cpu/$prefix-
$schritt".log"
sleep $wartezeit
107 done

109 # Dummynet - L2
prefix=L2dummynet
111
# rebooten, damit sicher nix mehr geladen ist
113 ssh -l root 132.252.152.152 "reboot"
sleep 300
115
# Konfiguration setzen
117 ssh -l root 132.252.152.152 "insmod /usr/local/src/Development/ipfw_linux
-20090724/dummynet/ipfw_mod.ko"
ssh -l root 132.252.152.152 "ipfw pipe 4 config bw 100Mbit/s queue 100"
119 ssh -l root 132.252.152.152 "ipfw add pipe 4 src-ether 00:01:02:df:4d:db
in"
ssh -l root 132.252.152.152 "ipfw pipe 5 config bw 100Mbit/s queue 100"
121 ssh -l root 132.252.152.152 "ipfw add pipe 5 dst-ether 00:01:02:df:4d:db
out"

123 # nun für jedes Mbit Messung durchführen
for schritt in {1..75}; do
125 echo -e -n $schritt" MBit/s:\t" >>cpu/$prefix-nuttcp.log
bandbreite=$(echo "scale=2; ($schritt * 1024 * $payload) / ($payload
+28+14)" | bc -l)k
127 ssh -l root 132.252.152.150 "nuttcp -u -R$bandbreite -
T$experimentdauer -l $payload 172.23.2.3" >>cpu/$prefix-nuttcp.log
&
sleep 5
129 ssh -l root 132.252.152.152 "vmstat -n 1 $messdauer" > cpu/$prefix-
$schritt".log"
sleep $wartezeit
131 done

```

Auswertung der Daten: Für die Auswertung wird awk genutzt.

```

1 basis=$1
  for bw in {1..75}; do
3     echo -n -e "$bw \t"
      tail -120 cpu/$basis-$bw.log | head -n 100 | cut -b 68- | awk '{
5         samples += 1
          user += $1
7         sys += $2
          idle += $3
9         }
      END {
11        CONVMT="%.6g"
          print user/samples/100"\t"sys/samples/100"\t"idle/samples/100
13        }'
  done

```

Literatur

- [1] Xen Community, 08 2007. URL <http://xen.xensource.com/>.
- [2] IPFW, 10 2009. URL <http://www.freebsd.org/doc/en/books/handbook/firewalls-ipfw.html>.
- [3] MTR, November 2009. URL <http://www.bitwizard.nl/mtr/>.
- [4] netfilter/iptables project homepage - The netfilter.org project, 10 2009. URL <http://www.netfilter.org/>.
- [5] NIST Net Home Page, 10 2009. URL <http://snad.ncsl.nist.gov/nistnet/>.
- [6] PF: The OpenBSD Packet Filter, 10 2009. URL <http://www.openbsd.org/faq/pf/>.
- [7] Iperf, November 2009. URL <http://iperf.sourceforge.net/>.
- [8] W. Almesberger. Linux Network Traffic Control - Implementation Overview. In *Proceedings of 5th Annual Linux Expo*, pages 153–164, Mai 1999.
- [9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003.
- [10] M. Carbone and L. Rizzo. DummyNet revisited. Technical report, Universität von Pisa, 5 2009.
- [11] M. Carbone and L. Rizzo. Adding emulation to Planetlab nodes. Technical report, Universität von Pisa, 06 2009.
- [12] M. Carson and D. Santay. NIST Net: A Linux-based Network Emulation Tool. *Computer Communication Review*, 33:2003, 2003.
- [13] D. Culler, T. Roscoe, L. Peterson, L. Peterson, T. Anderson, and T. Anderson. A blueprint for introducing disruptive technology into the internet. 2002.
- [14] J. Eklund. Evaluation of Emulab as Experimental Platform by Comparing TCP and SCTP . Master's thesis, Karlstad University, Juni 2004.
- [15] B. Fink. Manpage of NUTTCP, March 2007. URL <http://www.lcp.nrl.navy.mil/nuttcp/nuttcp.html>.
- [16] T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. B. Schroeder, J. Spaans, and P. Larroy. Classful Queueing Disciplines, Mai 2010. URL <http://lartc.org/howto/lartc.qdisc.classful.html>.

- [17] S. Hemminger. Network Emulation with NetEm. In *Linux Conf Au*, April 2005. URL <http://linux-net.osdl.org/index.php/Netem>.
- [18] B. Hubert. Linux Advanced Routing & Traffic Control HOWTO, Oktober 2009. URL <http://lartc.org/>.
- [19] G. Kurtsov. layer2 dummysnet, March 2009. URL <http://blogs.freebsdish.org/gleb/2009/03/24/layer2-dummysnet/>.
- [20] S. Lahde. Nistnet bugs, 10 2009. URL <http://www.ibr.cs.tu-bs.de/kb/nistnet.html>.
- [21] Linux Foundation. Net:IFB - The Linux Foundation, Oktober 2009. URL <http://www.linuxfoundation.org/en/Net:IFB>.
- [22] Linux Foundation. Net:Netem, 10 2009. URL <http://www.linuxfoundation.org/en/Net:Netem>.
- [23] I. Linux Kernel Organization. The Linux Kernel Archives, Mai 2010. URL <http://kernel.org/>.
- [24] J. D. C. Little. A proof for the queuing formula: $L = \lambda W$. *Operations Research*, 9(3):383–387, 1961. ISSN 0030364X. doi: 10.2307/167570. URL <http://dx.doi.org/10.2307/167570>.
- [25] M. Muuss. The Story of the TTCP Program, November 2009. URL <http://ftp.arl.army.mil/~mike/ttcp.html>.
- [26] I. Pratt. Xen Virtualization, 05 2005.
- [27] T. F. Project. The FreeBSD Project, Mai 2010. URL <http://www.freebsd.org/>.
- [28] L. Rizzo. Dummysnet home page, 10 2009. URL <http://info.iet.unipi.it/~luigi/dummysnet/>.
- [29] S. Salsano, F. Ludovici, and A. Ordine. Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel . Technical report, University of Rome “Tor Vergata”, Oktober 2009. URL <http://netgroup.uniroma2.it/twiki/bin/view.cgi/Main/NetEm2>.
- [30] B. D. Schuymer. ebttables, 10 2009. URL <http://ebtables.sourceforge.net/>.