HOCHSCHULE FÜR
TECHNIK UND WIRTSCHAFT
DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Diplomarbeit
im Studiengang:
Informatik

Entwicklung eines Messsystems für Netzwerkkomponenten mit 10 GBit/s auf OSI-Schicht 3

Peter Schumann

Betreuer	:	Prof. Dr.-Ing. J. Vogt, HTW Dresden
	:	Dipl.-Ing. R. Menzel, secunet Security Networks AG
Eingereicht am	:	18.10.2016

Inhaltsverzeichnis

Abkürzungsverzeichnis V

Abbildungsverzeichnis VII

Tabellenverzeichnis VII

Listingverzeichnis IX

1 Einleitung 1
- 1.1 Motivation . . . 1
- 1.2 Beschreibung der Aufgabenstellung . . . 2
- 1.3 Gliederung der Arbeit . . . 2

2 Grundlagen 4
- 2.1 Stand der Technik . . . 4
- 2.2 Der Ethernet Standard . . . 7
- 2.3 Leistungsbewertung nach RFC2544 . . . 9
  - 2.3.1 Definierte Tests . . . 10
  - 2.3.2 Format der Testpakete . . . 12
  - 2.3.3 Anforderungen an das Messsystem . . . 13
  - 2.3.4 Praxisrelevanz der Messmethoden . . . 14
- 2.4 Definition des IMIX Datenstroms . . . 14
- 2.5 Architektur der Intel Xeon E5-2600 v3 Plattform . . . 15
  - 2.5.1 Aufbau der Plattform . . . 15
  - 2.5.2 Besonderheiten der CPU . . . 16
- 2.6 Funktionsweise moderner Ethernet Chips am Beispiel des Intel 82599EB 17
  - 2.6.1 Aufbau der Funktionseinheiten . . . 18
  - 2.6.2 Hardwarezeitstempel . . . 20
- 2.7 Paketverarbeitung mit spezialisierter Software . . . 21
  - 2.7.1 Das netmap API . . . 21
  - 2.7.2 Data Plane Development Kit (DPDK) . . . 22

3 Entwurf und Architektur des Messsystems 25
3.1 Anforderungen . . . 25
3.2 Auswahl der Software und Hardware . . . 26
3.2.1 MoonGen . . . 26
3.2.2 Aufwand der Implementierung . . . 28
3.2.3 Auswahl der Hardware . . . 28
3.3 Software Architektur . . . 30
3.3.1 Das Stream Konzept . . . 31
3.3.2 Datenpfad . . . 32
3.3.3 Auswertung der Sequenznummern . . . 33
3.3.4 Konfiguration des Messsystems . . . 36
3.3.5 Beschreibung der Ablaufsteuerung . . . 36

4 Umsetzung mit MoonGen 39
4.1 Erweiterung der Stream Klasse . . . 39
4.2 NUMA Konzept im Datenpfad . . . 39
4.3 Der Datenpfad . . . 41
4.3.1 Funktionsweise der TX Tasks . . . 41
4.3.2 Funktionsweise der RX Task . . . 43
4.4 Latenzmessung . . . 46
4.5 Suchverfahren . . . 47
4.6 Fehler der verwendeten Softwarekomponenten und deren Lösung . . . 49
4.7 Dokumentation der Messungen . . . 51

5 Messungen und Messaufbauten 52
5.1 Beschreibung der Referenzmesssysteme . . . 52
5.1.1 Spirent Testcenter N4U . . . 52
5.1.2 EXFO FTB-400 . . . 53
5.2 Messaufbauten . . . 54
5.2.1 VPN . . . 54
5.2.2 Direktverbindung . . . 56

6 Auswertung der Messergebnisse 58
6.1 Leistungsfähigkeit der Paketverarbeitung . . . 58
6.2 Genauigkeit der TX Raten Begrenzung . . . 59
6.3 Auswertung der Vergleichsmessungen . . . 62
6.3.1 Auswertung der Durchsatz Tests . . . 62
6.3.2 Auswertung der Latenz Tests . . . 64
6.3.3 Auswertung der BackToBack-Tests . . . 67
6.3.4 Auswertung der IMIX-Tests . . . 68

7 Zusammenfassung und Ausblick 70

Quellenverzeichnis 73

Anhang 77

A Bewertungsmatrix für netmap, DPDK und MoonGen 77

B Hardware Komponenten der Device Under Test (DuT) 78

C IMIX Parameter 79

D Messergebnisse 80

Abkürzungsverzeichnis

API Application Programming Interface . . . 2
ARP Address Resolution Protocol . . . 10
ASIC Application Specific Integrated Circuit . . . 5
CSMA/CD Carrier Sense Multiple Access/Collision Detection . . . 7
DCA Direct Cache Access . . . 17
DMA Direct Memory Access . . . 18
DPDK Data Plane Development Kit . . . II
DuT Device Under Test . . . IV
EAL Environment Abstraction Layer . . . 22
FDDI Fiber Distributed Data Interface . . . 9
FFI Fast Function Interface . . . 44
FPGA Field Programmable Gate Array . . . 5
IO Input/Output . . . 2
IPsec Internet Protocol Security . . . 4
IEEE Institute of Electrical and Electronics Engineers . . . 7
IETF Internet Engineering Task Force . . . 2
IFG Inter Frame Gap . . . 6
IMIX Internet Mix . . . 14
IP Internet Protocol . . . 9
ITU-T International Telecommunication Union - Telecommunication Standardization Sector . . . 4
MIFS Minimun Inter Frame Spacing . . . 19
MMIO Memory Mapped Input Output . . . 18
Mpps Millionen Paket Pro Sekunde . . . 5
NUMA Non Uniform Memory Access . . . 16
OSI-Model Open System Interconnection Model
PMD Poll Mode Driver . . . 22
PTP Precision Time Protocol . . . 20
QSFP+ Quad Small Form-factor Pluggable . . . 52
QPI Quick Path Interconnect . . . 15
RFC Request For Comment . . . 4
SFP Small Form-factor Pluggable . . . 53

SFP+ Small Form-factor Pluggable Plus 28
SLA Service Level Aggrement 4
TRS Transmit Rate Scheduler 19
TSC Time Stamp Counter 43
UDP User Datagram Protocol 6

Abbildungsverzeichnis

1 Ethernet Rahmen . . . 8
2 Messaufbau RFC2544 . . . 10
3 RFC2544 Rahmenformat - Ethernet IP UDP . . . 13
4 Intel Haswell Chip Ausbaustufen . . . 16
5 TRS Auflösung in Abhängigkeit der relativen Datenrate und Rahmenlänge . . . 20
6 DPDK Mempool Cache . . . 24

7 Systemarchitektur Dell PowerEdge R530 . . . 30
8 Softwarearchitektur des Messsystem . . . 31
9 Datenpfad Schema . . . 32
10 Synchronisierung von RX und TX Tasks . . . 38

11 NUMA gerechte Task Zuorndnung . . . 40

12 Messaufbau VPN . . . 54
13 Messaufbau Direktverbindung . . . 56

14 TRS Fehler in Abhängigkeit der Layer 1 Datenrate und Rahmenlänge . 60
15 TRS Fehler in Abhängigkeit der Layer 1 Datenrate und Rahmenlänge ($\leq$88 Byte) . . . 61
16 TRS Fehler in Abhängigkeit der Layer 1 Datenrate und Rahmenlänge ($\geq$88 Byte) . . . 61
17 Messbereich Datenrate für nettest . . . 62
18 Messergebnisse: Datenraten - DuT 1, Linux . . . 63
19 Messergebnisse: Frameloss - DuT 1, Linux, nettest . . . 66
20 Frameloss Messung DuT1: Abweichung . . . 67

Tabellenverzeichnis

1 Übersicht gängiger Ethernet Standards und deren Kennwerte 9
2 Beispiel IMIX Verteilung 14
3 Beispiel IMIX Schedule 15
4 Intel 82599EB: Timer Granularität 21

5 Hardware des Entwicklungssystems 29

6 Durchgeführte Messungen für Aufbau VPN 55
7 Testparameter: RFC2544 und IMIX 55
8 Schnittstellen und Sockel Kombinationen für Aufbau "Direktverbindung" 56
9 Rahmenlängen für TRS Messung 57

10 Direktverbindung - maximale Datenrate / % Linerate 58
11 Messergebnisse: Durchsatz - DuT 1 Linux 64
12 Auswertung: Durchsatz Test(1024,1518), Mittelwert $\overline{\mathbf{X}}$ und Standardabweichung $\boldsymbol{\sigma}$, DuT 1 64
13 Messergebnisse: Latenz - DuT 1 64
14 Auswertung: Latenz Test, Differenz nettest zu Spirent, DuT 1 65
15 Messergebnisse: Back to Back - DuT 2, pfSense 68
16 Messergebnisse: Back to Back - DuT 1 68
17 Messergebnisse: IMIX DuT 1 68

18 Bertungsmatrix für netmap, DPDK und MoonGen 77

19 Hardware der DuT Systeme 78

20 Testparameter: IMIX Zusammensetzung 79

21 Messergebnisse: Durchsatz, DuT 1, Linux 80
22 Messergebnisse: Latenz, DuT 1, Linux 80
23 Messergebnisse: BackToBack, DuT 1, Linux 80
24 Auswertung: DuT 1, Linux, Frameloss Differenz $P_{loss,Spirent} - P_{loss,nettest}$ 81
25 Messergebnisse: IMIX, DuT 1, Linux 81
26 Messergebnisse: Durchsatz, DuT 1, Linux - $L = \{1024, 1518\}$ 82

27 Messergebnisse: Durchsatz, DuT 2, Linux 83
28 Messergebnisse: Latenz, DuT 2, Linux 83
29 Messergebnisse: BackToBack, DuT 2, Linux 83
30 Messergebnisse: IMIX, DuT 2, Linux 83
31 Messergebnisse: Durchsatz, DuT 2, pfSense 84
32 Messergebnisse: Latenz, DuT 2, pfSense 84
33 Messergebnisse: BackToBack, DuT 2, pfSense 84
34 Messergebnisse: IMIX, DuT 2, pfSense 84

Listingverzeichnis

1 Sequenznummern Auswertung - Naiv 34
2 Sequenznummern Auswertung - Erweitert 35

3 Sequenznummern Auswertung - Bitfeld Operationen 44
4 Sequenznummern Auswertung - Paketverlust 45
5 Sequenznummern Auswertung - Paketverlust IMIX C Teil 45
6 Sequenznummern Auswertung - Paketverlust IMIX Lua Teil 46
7 Suchalgorithmus: binäre Suche mit Grenzwert 48

1 Einleitung

Die zunehmende Vernetzung der gesamten Welt erfordert immer schnellere Datenübertragung um dem Bedarf an Bandbreite gerecht zu werden. Das Datenaufkommen am DE-CIX Internet Knoten in Frankfurt hat sich im Zeitraum von 2012 bis 2016 etwa verdreifacht[1]. Allein durch die Popularität von Streaming-Diensten wie Youtube und Netflix wird dieser Trend auch weiterhin bestand haben. Um auch in Zukunft ausreichend Übertragungskapazität bieten zu können, ist eine stetige Weiterentwicklung der Netzwerkkomponenten in allen Einsatzbereichen, vom DSL Modem bis zu Core Routern nötig. Für die Entwicklung neuer Netzwerkkomponenten werden Messsysteme benötigt um die Leistungsfähigkeit und Korrektheit nachzuweisen.

Die Entwicklung eben dieser Messsystems ist auf Grund der speziellen Hardware und durch relativ geringe Stückzahlen sehr teuer. Für einen aktuelles Messsystem muss mit Kosten im sechs stelligen Bereich kalkuliert werden. Dadurch ist der breite Einsatz, beispielsweise für entwicklungsbegleitende Tests, mit erheblichem finanziellem Aufwand verbunden. In dieser Arbeit wird ein Messsystem für Netzwerkkomponenten der OSI Schicht-3 vorgestellt und evaluiert, dass für den Bruchteil der Kosten realisiert werden kann. Durch die Verwendung offener Schnittstellen und Open Source Software ist außerdem eine Erweiterung und Anpassung an spezielle Anwendungen sehr einfach möglich.

1.1 Motivation

Kommerzielle Messsysteme für Netzwerkkomponenten basieren auf speziell entwickelter Hardware, um die hohen Anforderungen an Zeitverhalten und Geschwindigkeit erfüllen zu können. Die Leistungsfähigkeit aktueller Computersysteme und deren Netzwerkschnittellen hat ein Niveau erreicht, das es ermöglicht, Messsysteme auf dieser Basis zu entwickeln. Durch die Verwendung von Standard-Komponenten

[1] siehe Grafik auf [DEC]

können die Kosten für ein Messsystem erheblich reduziert werden. Gängige Messsysteme kosten etwa das dreißigfache eines leistungsfähigen Serversystems. Für ein Messsystem auf Basis eines Standard X86 Rechnersystems wird neben der Hardware auch Software benötigt. Nahezu alle existierenden Programme haben architekturbedingte Schwächen in der Leistungsfähigkeit und können die Möglichkeiten der Hardware deshalb nur teilweise ausnutzen. Außerdem entsprechen die Testverfahren keinem Standard, daher sind die Ergebnisse nur eingeschränkt mit denen anderer Messsysteme vergleichbar. Durch Nutzung neuer Paketverarbeitungstechnologien ist es möglich, das volle Potenzial der Standard x86 Hardware auszuschöpfen und damit ein Messsystem zu entwickeln, dass sich mit kostenintensiven Spezialsystemen messen kann.

1.2 Beschreibung der Aufgabenstellung

Ziel der Arbeit ist es, ein Messsystem für Netzwerkkomponenten auf OSI-Schicht 3, zu entwickeln, welches auf Standard Server Hardware basiert und die Leistungsfähigkeit bietet, 10GBit/s Ethernet Netzwerkkomponenten zu vermessen. Als Messverfahren soll der Internet Engineering Task Force (IETF) Standard RFC2544 [BM1999] implementiert werden. Zusätzlich ist gefordert, dass eine Durchsatz-Messung mit realistischen Rahmenlängenverteilungen möglich ist. Das zu entwickelnde Messsystem soll im Rahmen der Evaluation mit am Markt verfügbaren Messsystemen verglichen und bewertet werden.

1.3 Gliederung der Arbeit

Die Arbeit gliedert sich in die fünf folgenden Abschnitte. Im Abschnitt Grundlagen wird ein Überblick der aktuellen Entwicklungen in den Bereichen kommerzielle Messsysteme sowie Paketverarbeitung in Rechnersystemen gegeben. Darauf folgen Ausführungen zum Ethernet Protokoll sowie dem RFC2544 Standard. Im Anschluss werden wichtige Eigenschaften und Architekturmerkmale eines aktuellen Serversystems sowie dessen Netzwerkschnittstellen erklärt. Der letzte Abschnitt im Kapitel Grundlagen ist eine Einführung in die Funktionsweise verbreiteter Fast Packet Input/Output (IO) Application Programming Interfaces (APIs). Im darauf folgenden Kapitel wird die Architektur des entwickelten Messsystems, basierend auf den Anforderungen, beschrieben und die wesentlichen Softwarekomponenten, Konzepte

und Algorithmen vorgestellt. Auf Grundlage der Architektur werden die Hardwarekomponenten für das Messsystem ausgewählt. Anschließend behandelt das Kapitel „Umsetzung mit MoonGen“ die Implementierungsdetails der wesentlichen Komponenten des Messsystems und erklärt die Lösungen zu identifizierten Problemen. Die zwei folgenden Kapitel beschäftigen sich mit der Evaluierung des entwickelten Messsystems: Im Kapitel „Messungen“ werden die Referenzmessgeräte, Messaufbauten und alle durchgeführten Messungen erklärt. Auf Grundlage der Messergebnisse erfolgt in Kapitel 6 die Auswertung und Diskussion hinsichtlich der Plausibilität und Qualität der Messwerte sowie der Leistungsfähigkeit des entwickelten Messsystems. Abschließend werden Ansatzpunkte für die Weiterentwicklung des Messsystems vorgestellt.

2 Grundlagen

In diesem Kapitel werden die nötigen Grundlagen, für die Entwicklung eines Messsystems für OSI-Schicht 3 Netzwerkkomponenten, vermittelt. Im ersten Abschnitt wird der aktuelle Stand der Technik beschrieben und die prinzipielle Funktionsweise von Messsystemen für paketverarbeitende Netzwerke. Im zweiten Abschnitt wird die meistgenutzte Netzwerktechnologie, das Ethernet, vorgestellt. Darauf folgt im dritten Abschnitt die Beschreibung des RFC2544 Standards. In den anschließenden Abschnitten wird auf die Architektur, Funktionsweise sowie Besonderheiten der aktuellen Intel Xeon Plattform am Beispiel der Xeon 26xx v3 Serie sowie des Intel 82599EB Netzwerk Chips eingegangen. Ein gutes Verständnis der Hardware ist besonders für die Entwicklung hoch performanter Software unabdingbar. Der sechste Abschnitt stellt zwei Ansätze vor, die die Verarbeitung sehr hoher Paketraten sowie die Entwicklung von genauen Messsystemen, durch Ausnutzung spezieller Funktionen der Netzwerkhardware, möglich machen.

2.1 Stand der Technik

Ein Messsystem für Netzwerkkomponenten ermöglicht es, ausgewählte Eigenschaften und Leistungsmerkmale zu bestimmen. Je nach Einsatzbereich und verlangten Messwerten gibt es verschiedene Messverfahren. Diese Arbeit beschäftigt sich mit den für die OSI-Schicht 3 relevanten Verfahren, insbesondere bezogen auf Internet Protocol Security (IPsec) Gateways. Hier sind vor allem die erreichbaren Übertragungsgeschwindigkeiten, die Verzögerung sowie die Integrität des Datenstroms von Bedeutung. Integrität bezieht sich hierbei auf die korrekte Reihenfolge sowie den korrekten Inhalt jedes einzelnen Datenpakets. Das gängigste Messverfahren wird im Request For Comment (RFC) 2544 der IETF beschrieben und gilt als Standardverfahren für die Charakterisierung der Leistungsfähigkeit von Netzwerkkomponenten. Ein weiteres bekanntes Testverfahren wird von der International Telecommunication Union - Telecommunication Standardization Sector (ITU-T), unter der Bezeichnung *Y.1564* beziehungsweise *EtherSAM*, standardisiert. Die dort beschrieben Messverfahren dienen der Untersuchung von Service Level Aggrement (SLA). In

einem SLA wird festgelegt wie viel Übertragungskapazität bestimmten Diensten (OSI-Schicht 4 und 5) zugesichert wird und ist daher für diese Arbeit nicht relevant.

Grundsätzlich besteht ein Messsystem aus zwei Komponenten: Die erste Komponente erzeugt Datenpakete mit definierten Eigenschaften. Die Aufgabe der zweiten Komponente ist es, diese Pakete, nach dem sie die Messstrecke passiert haben, auszuwerten. Die Genauigkeit der Erzeugung und Auswertung bestimmen maßgeblich die Qualität der Messergebnisse. Aber auch die Leistungsfähigkeit ist ein wichtiger Faktor. Kann ein Messsystem nicht genügend Pakete generieren, um die zu messende Komponente an ihre Leistungsgrenze zu bringen, ist es nicht möglich, diese zu ermitteln. Um eine 10GBit/s Ethernet Verbindung zu saturieren sind bei kleinster erlaubter Rahmengröße von 64 Byte, 14,88 Millionen Paket Pro Sekunde (Mpps)[1] nötig. Dies entspricht einer Rahmendauer von 67,2 ns, bei einer CPU mit 2,6 GHz Taktfrequenz stehen damit nur etwa 174 Zyklen[2] zur Verfügung, in denen das Paket erzeugt werden muss. Das genaue Rahmenformat sowie die Berechnungen werden im nächsten Abschnitt vorgestellt.

Die meisten im Handel erhältlichen Messsysteme realisieren diese Funktionen mit speziell dafür entwickelter Hardware. Durch die Verwendung von Field Programmable Gate Arrays (FPGAs) können problemlos und zuverlässig sehr hohe Daten- und Paketeraten erreicht werden. Außerdem ist dadurch eine sehr genaue Messung der Paketlaufzeiten im Bereich einiger Nanosekunden möglich. Diese Messsysteme sind dafür ausgelegt, immer die maximale Übertragungsrate der unterstützten Netzwerktechnologie erzielen zu können. Dadurch eignen sie sich insbesondere um Netzwerkkomponenten der OSI-Schichten 1 und 2, wie Switch Application Specific Integrated Circuit (ASIC) und Medienkonverter, zu untersuchen. Der hohe Entwicklungsaufwand sowie die geringen Stückzahlen machen diese FPGA basierten Messsysteme sehr teuer. Für ein Messsystem mit zwei 10 GBit/s Schnittstellen ist ein mittlerer 5-stelliger Betrag durchaus üblich. Werden höher Datenraten oder mehr Schnittstellen benötigt ist mit Anschaffungskosten ab 100 000 € zu rechnen.

Außerdem gibt es Messsysteme, die auf Standard Computern und deren Betriebssystemen aufsetzen, wie das sehr verbreitete Programm *iperf* [3]. Da es sich hierbei um Open Source Software handelt, entstehen nur die Beschaffungskosten für die Hardware. Somit ist es gegenüber kommerziellen Systemen, die auf Spezialhardware

[1] $14,88 Mpps \approx \frac{10000 MBit/s}{((64 Byte + 20 Byte) * 8 \frac{Bit}{Byte})}$

[2] 174 Zyklen $\approx$ 67,2 ns * 2,6 GHz

[3] https://iperf.fr/

beruhen, erheblich günstiger. Die Erzeugung und Auswertung der Datenpakete wird hier vollständig per Software bewerkstelligt. Daraus ergeben sich Vor- aber auch Nachteile. Da als Basis ein Standard Betriebssystem, wie Debian/GNU Linux, verwendet wird, kann jedes Computersystem, auf dem dieses lauffähig ist und über entsprechende Netzwerkschnittellen verfügt, als Messsystem verwendet werden. Somit ist es gegenüber FPGA basierten Systemen relativ kostengünstig möglich, viele Messsysteme zu betreiben. Die Leistungsfähigkeit der Paketverarbeitung hängt aber stark von den verwendeten Komponenten, wie beispielsweise der CPU,den Netzwerk Chips sowie der Konfiguration des Computersystems ab. Daraus ergibt sich eine Einschränkung der maximal erzielbaren Paket- und Datenrate, die sich für jede neue Kombination aus Hardware, Konfiguration und Software ändern kann.

Eine weitere Schwäche der Software Lösung besteht darin, dass es mit diesem Ansatz konzeptionell nicht möglich ist, sehr hohe Paketraten zu erzeugen. Die Ursache dafür ist, dass für jedes Datenpaket ein Systemruf nötig ist, um es dem Betriebssystem zu übergeben. Anschließend durchläuft das Datenpaket alle Ebenen des Netzwerkprotokollstapels und wird letztendlich dem Treiber der Netzwerkschnittstelle zum Versenden übergeben. In [RIZ2012, Abschnitt 2.3] hat Rizzo dass Verschicken eines 64 byte User Datagram Protocol (UDP) Pakets unter FreeBSD hinsichtlich der Verarbeitungszeit untersucht. Allein der *sendto()* Systemruf benötigt 96 ns, bis das Paket die Netzwerkschnittelle verlassen hat vergingen 950 ns. Für das saturieren einer 10GBit/s Ethernet Verbindung bei dieser Paketgröße muss jedoch alle 67,2 ns ein Paket gesendet werden. Daher ist es auf diesem Weg nicht möglich, hohe Datenraten bei kleinen Paketgrößen zu erzeugen. Mit der von Rizzo ermittelten Verarbeitungszeit von 950 ns ist theoretisch eine maximale Paketrate von 1,0526 Mpps möglich. Um die geforderten 14,88 Mpps zu erreichen ist ein anderer Ansatz nötig.

Eine weitere Schwierigkeit bei der Paketerzeugung in Software ist die Steuerung der Sendegeschwindigkeit. Wenn nicht mit der höchstmöglichen Datenrate gesendet werden soll ist es erforderlich, dass zwischen den einzelnen Datenpaketen eine Verzögerung eingefügt wird. Diese wird auch als Inter Frame Gap (IFG) bezeichnet. Um 50 % Auslastung auf einer 10GBit/s Ethernet Verbindung zu erzeugen, muss auf jedes Paket eine ebenso lange Pause folgen. Bei einer Rahmenlänge von 64 Byte bedeutet das, dass zwischen dem Senden der Pakete 67,2 ns gewartet werden muss. Der Linux Kernel bietet mehrere Möglichkeiten um Zeitverzögerungen im Programmfluss zu realisieren, deren Auflösung liegt jedoch bestenfalls im Bereich mehrerer µS [GNU, vgl. High-resolution timers] und ist somit viel zu grob. Allein der Aufwand für einen Systemruf übersteigt die nötige Wartezeit.

Der Einsatz Software-basierter Messsysteme, die auf dem Netzwerkstack eines Betriebssystems aufsetzen, ist daher nur sinnvoll, wenn das Ziel die Erzeugung hoher

Datenraten bei geringen Paketraten und daraus resultierende großen Rahmenlängen ist. Wenn jedoch auch hohe Paketraten gefordert sind, muss der konventionelle Netzwerkstack umgangen werden. Im Abschnitt 2.7 werden zwei Lösungen vorgestellt, die es ermöglichen, maximale Paketraten auf Standard Hardware zu erreichen.

2.2 Der Ethernet Standard

Ethernet ist heute der meistgenutzte Standard für die Vernetzung von Rechnersystemen. Es hat sich gegenüber anderen Technologien durchgesetzt weil es flexibel, wartungsfreundlich und erweiterbar sowie kostengünstig ist, vgl. [TAN2003]. Es entstand ursprünglich in den 1970er Jahren am Xerox Palo Alto Research Center. Die Daten werden in Rahmen verpackt und über ein gemeinsames Übertragungsmedium gesendet. Das heute immer noch verwendete Rahmenformat ist im Institute of Electrical and Electronics Engineers (IEEE) Standard 802.3 von 1983 definiert. Im IEEE 802.3 wird neben dem Rahmenformat auch die Elektrische oder Optische Übertragung vorgeschrieben. Ursprünglich wurde für alle Teilnehmer eines Ethernet Netzwerks das selbe Übertragungsmedium verwendet. Als Zugriffsverfahren wurde Carrier Sense Multiple Access/Collision Detection (CSMA/CD) eingesetzt. Für die Kollisionserkennung des CSMA/CD Verfahrens ist es nötig, dass die Datenrahmen eine bestimmte Mindestlänge nicht unterschreiten. Diese Länge wird durch die maximale räumliche Ausdehnung des physischen Netzes bestimmt. Für die heute relevanten Ethernet Standards spielt die Kollisionserkennung jedoch keine Rolle mehr, da die meisten Netze im Vollduplex-Modus betrieben werden und durch den Einsatz von Switches Punkt zu Punkt Verbindungen zustande kommen. Dadurch ist die Kollisionsdomäne nicht mehr vorhanden, weil bei nur zwei Teilnehmern über eine Vollduplexverbindung, keine Kollision auftreten kann. Es wird jedoch auch weiterhin eine Mindestrahmenlänge (L_{min}) von 64 Byte gefordert.

Aufbau des Ethernet Rahmen

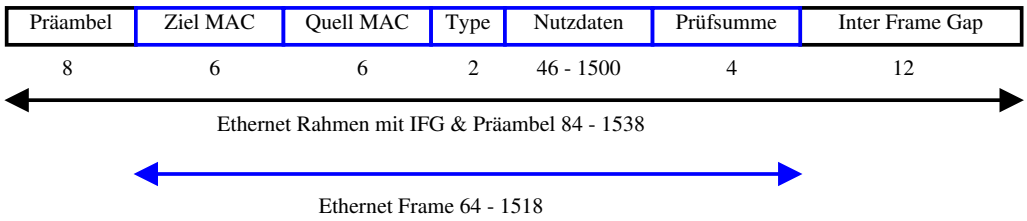


Abbildung 1: Ethernet Rahmen, alle Zahlenwerte in Byte

Ein Ethernet Rahmen kann eine Länge L von 64 bis zu 1518 Bytes haben. In **Abbildung 1** ist die Länge und Reihenfolge der einzelnen Felder dargestellt. Alle Zahlen sind in Byte angegeben. Vor jedem Rahmen wird eine Präambel mit 8 Byte Länge eingefügt, auf das Ende des Rahmen folgt das IFG mit 12 Byte Länge. Somit benötigt ein Ethernet Rahmen auf der OSI-Schicht 1 immer 20 Byte (L_{over}) extra. Die gesamte Länge, die ein Rahmen auf dem Medium benötigt, wird als L_{wire} bezeichnet. Damit lässt sich die Datenrate der OSI-Schicht 2 (D_{eth}) mit Gleichung (1) berechnen.

$$D_{eth} = D_{wire} * \frac{L}{L + L_{over}} \tag{1}$$

Anhand der Mindestrahmenlänge L_{min} und der Datenübertragungsrate D_{link} lässt sich mit **Formel** (2) die maximale Paketrate P_{max} bestimmen. Allgemein gilt die **Gleichung** (3) zur Bestimmung der Paketrate für beliebige L. Mit **Gleichung** (4) erhält man die Paketdauer T_{frame} aus der Paketrate P.

$$P_{max} = \frac{D_{link}}{(L_{min} + L_{over}) * 8\frac{Bit}{Byte}} \tag{2}$$

$$P = \frac{D_{link}}{(L + L_{over}) * 8\frac{Bit}{Byte}} \tag{3}$$

$$T_{frame} = \frac{1\ Paket}{P} \tag{4}$$

Für die gängigen Ethernet Geschwindigkeiten sind die Kennwerte D_{link}, P_{max} und

T_{frame} für 64 Byte Rahmen in **Tabelle 1** aufgelistet. Die Datenrate D bezieht sich immer auf eine Richtung der Datenübertragung. Der Vollduplex Betrieb ist für Gigabit Ethernet und schnellere Varianten vorgeschrieben.

Tabelle 1: Übersicht gängiger Ethernet Standards und deren Kennwerte

Ethernet Typ	$\mathbf{D_{link}}$ **/ Mbit/s**	$\mathbf{P_{max}}$ **/ Mpps**	$\mathbf{T_{frame}}$ **/ ns**
100Base-T	100	0,1488	6720,00
1000Base-T	1000	1,4881	672,00
10GBASE-SR	10000	14,8810	67,20
40GBASE-SR4	40000	59,5238	16,80
100GBASE-SR4	100000	148,8095	6,7

2.3 Leistungsbewertung nach RFC2544

In diesem Abschnitt werden Methoden und Messwerte der RFC 2544 Standards ("Benchmarking Methodology for Network Interconnect Devices") erklärt. Dieses Standardverfahren wird von den meisten kommerziell erhältlichen Messsystemen für Netzwerkkomponenten implementiert. Durch das genormte Messverfahren können die Messwerte unterschiedlicher DuT direkt verglichen werden. Als Netzwerktechnologie wird hier ausschließlich auf Ethernet eingegangen. Es muss nicht zwingend Ethernet verwendet werden, aber durch die weite Verbreitung sind andere Verfahren wie TokenRing und Fiber Distributed Data Interface (FDDI) praktisch nicht relevant. Auf der Vermittlungsschicht wird das Internet Protocol (IP) mit einem UDP Header als Nutzdaten gefordert.

Das Messsystem wird als *Tester* bezeichnet und das zu vermessende Gerät als **DuT**. Die beschriebenen Tests haben gemein, dass ein Paketstrom mit genau definierten Eigenschaften in das DuT eingeleitet wird. Nach der Verarbeitung verlässt der Paketstrom das DuT und wird im *Tester* ausgewertet. Die Messgrößen sind in den einzelnen Tests definiert. Der Messaufbau ist in **Abbildung 2** dargestellt. Das DuT wird durch zwei Ethernet Verbindungen mit dem *Tester* verbunden (siehe [BM1999] Page 3, Fig. 1).

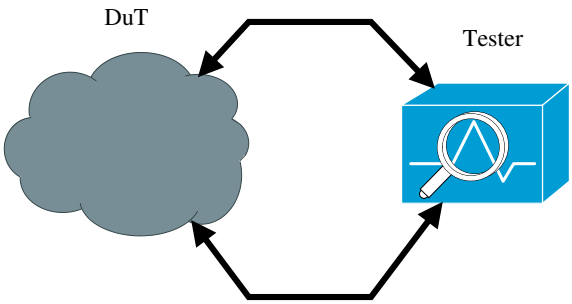


Abbildung 2: Messaufbau nach RFC2544

2.3.1 Definierte Tests

Im RFC2544 sind vier Tests definiert: Durchsatz, BackToBack, Latenz und Paketverlust. Jeder Test wird für eine festgelegte Reihe von Rahmengrößen durchgeführt. Der Standard fordert, dass die maximale und minimale Rahmengröße des Mediums sowie mindestens drei weitere enthalten sein müssen. Es wird empfohlen, folgende Rahmenlängen zu verwenden: 64, 128, 256, 512, 1024, 1280 und 1518 Byte (siehe [BM1999, Appendix B]).

Ein Test besteht in der Regel aus mehreren Durchläufen pro Rahmenlänge. Die Durchläufe, im RFC 2544 *Trail* genannt, haben für alle Tests den gleichen Ablauf. Ein *Trail* wird in vier Phasen unterteilt:

1. Aufwärm-Rahmen senden
2. den Test ausführen
3. zwei Sekunden auf ausstehende Rahmen warten
4. fünf Sekunden auf die Erholung des DuT warten

Phase 1 dient dem Initialisieren der Address Resolution Protocol (ARP) Tabellen des DuT. In Phase 2 werden die für den Test nötigen Pakete gesendet und die empfangenen Pakete ausgewertet. Diese Phase sollte nicht kürzer als 60 Sekunden sein. Nach Ablauf der Zeit wird das Senden von Paketen eingestellt. Phase 3 hat den Zweck, noch ausstehende Rahmen, nach dem Ende der 2. Phase, einzusammeln. Die 4. und letzte Phase eines Testdurchlaufs soll sicherstellen, dass das DuT wieder einen stabilen Betriebszustand erreicht.

Durchsatz-Test

Der Durchsatz-Test hat das Ziel, die maximale Datenrate $D_{max}(L)$, die das DuT bei einer fixen Rahmenlänge L verarbeiten kann, zu ermitteln. Dazu wird eine bestimmte Anzahl an Datenpaketen an das DuT gesendet. Werden weniger Pakete empfangen, wird die Datenrate reduziert, da offensichtlich Pakete verworfen worden sind. Stimmt die Anzahl der empfangenen Pakete mit der Anzahl der gesendeten überein, wird die Datenrate angehoben. Für das Ändern der Datenrate wird empfohlen, nach dem Algorithmus der binären Suche vorzugehen. Das Ergebnis des Durchsatz Tests ist die maximale Datenrate für jede Rahmengröße, die das DuT ohne Verluste weiterleiten kann.

Latenz-Test

Der Latenz-Test misst die Zeit, die ein Datenpaket benötigt, nach dem es durch den Tester versendet wurde (Zeitpunkt t_A), das DuT passiert und wieder von Tester empfangen wird (Zeitpunkt t_B). Die Latenz ΔT ergibt sich aus der Differenz $t_B - t_A$. Es ist gefordert, dass die im Durchsatz-Test ermittelte Datenrate auf das DuT wirkt. Der genaue Ablauf eines Test sieht vor, dass das DuT für mindestens 120 Sekunden mit $D_{max}(L)$ belastet wird. Nach 60 Sekunden (t_A) soll ein Rahmen mit einer Markierung versehen werden. Dies ermöglicht dem Empfänger genau diesen Rahmen zu erkennen und den Zeitpunkt t_B zu bestimmen. Dieser Vorgang muss mindestens zwanzig Mal wiederholt werden. Der Mittelwert aller Wiederholungen ist das Ergebnis der Latenz-Tests.

Paketverlust-Test

Für den Paketverlust-Test wird absteigend von der maximalen Senderate, mit einer Schrittweite von höchstens 10%, sukzessive jede Rate vermessen. Dazu wird, wie bei dem Durchsatz Test, eine bestimmte Anzahl Datenpakete gesendet (N_{tx}) und die Zahl der empfangenen Pakete (N_{rx}) ausgewertet. Für jede eingestellte Datenrate wird die Paketverlust-Rate P_{loss} nach der **Formel** (5) errechnet. Sobald bei einem Durchgang keine Verluste auftreten, kann mit der nächsten Rahmengröße fortgefahren werden. Das ist möglich, da angenommen wird, dass das DuT bei geringeren Datenraten (weniger Last) keine Pakete verwirft. Das Ergebnis dieses Tests wird in der Regel als Graph dargestellt, wobei P_{loss} auf der Y-Achse über die prozentua-

le Datenrate aufgetragen wird. Es wird für jede Rahmengröße ein solcher Graph erstellt.

$$P_{loss} = ((N_{tx} - N_{rx}) * 100)/N_{tx} \tag{5}$$

BackToBack-Test

Der BackToBack-Test soll ermitteln, wie viele Rahmen ein DuT verarbeiten kann, wenn diese mit dem minimalen Abstand versendet werden. Blöcke von Paketen, die mit minimalen Abstand gesendet werden, werden auch *Burst* genannt. Für einen Testdurchlauf wird ein Burst der Länge N_{burst} gesendet, falls alle Rahmen vom DuT weitergeleitet werden, wird N_{burst} vergrößert, anderenfalls verringert. Es bietet sich an, auch bei diesem Test, die binäre Suche zu verwenden.

2.3.2 Format der Testpakete

Als Paketformat wird ein Ethernet Rahmen, mit IPv4/UDP Paket als Nutzdaten, im RFC 2544 vorgeschrieben (siehe **Abbildung 3**). Damit IP Pakete über Ethernet übertragen werden können, muss die IP Adresse zu einer MAC Adresse aufgelöst werden. Dafür wird das ARP verwendet und muss daher unterstützt werden. Außerdem wird empfohlen, eine nicht näher beschriebene Sequenznummer in jedem Rahmen zu übertragen, um Fehler in der Reihenfolge der Rahmen sowie doppelte und fehlende Rahmen erkennen zu können.

Ein großer Anteil der Header-Felder sind statisch, bezogen auf einen Testlauf. Alle grau hinterlegten Felder erfordern es, unter bestimmten Umständen angepasst zu werden: Bei Änderung der Rahmenlänge müssen die Felder *TotalLength* und *UDPLength* angepasst werden, wodurch die Neuberechnung der Felder *IPChecksum* und *UDPChecksum* nötig wird. Falls eine Sequenznummer im UDP Payload Teil übertragen werden soll, ist die Neuberechnung des *UDPChecksum* Felds für jeden Rahmen nötig.

Die IPv4- und der UDP-Header benötigen gemeinsam 28 Byte, damit stehen, bei einer minimalen Ethernet Nutzdatenlänge von 46 Byte (siehe Abschnitt 2.2), mindestens 18 Byte für die Übertragung von Nutzdaten zur Verfügung.

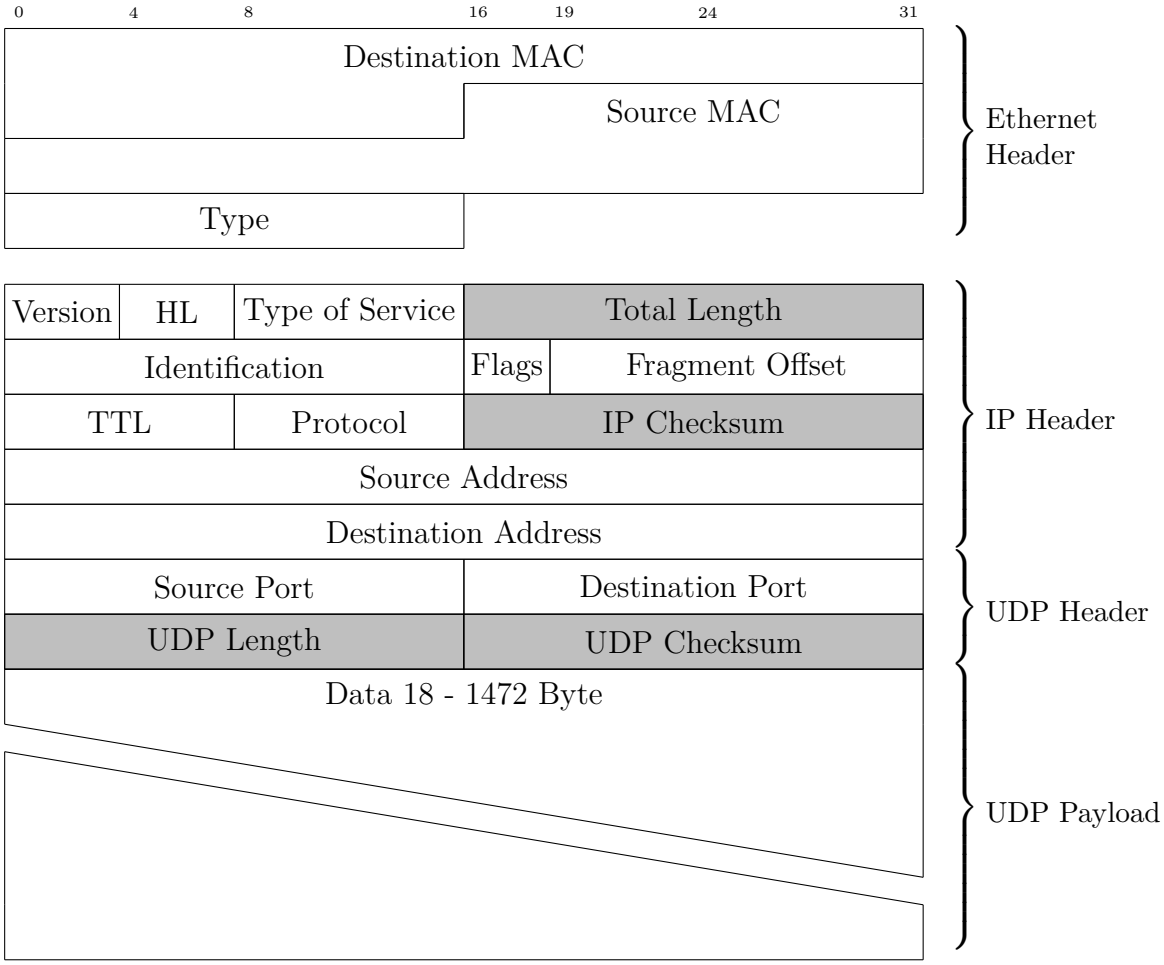


Abbildung 3: RFC2544 Rahmenformat - Ethernet IP UDP

2.3.3 Anforderungen an das Messsystem

Aus den Methoden und Vorgaben des RFC 2455 ergeben sich technische Anforderungen, die eine Implementierung erfüllen muss:

1. Das Messsystem muss ARP Funktionalität bieten
2. Senden definierter Pakete mit konstanter Datenrate
3. Empfangen der Pakete und Auswertung der Sequenznummern
4. Senden und Empfangen bei maximaler Paketrate

5. Zählen der gesendeten und empfangenen Rahmen

6. Messung der Paketlaufzeit

7. Generieren eines Messprotokolls

2.3.4 Praxisrelevanz der Messmethoden

Die Messergebnisse der im RFC 2544 beschriebenen Tests sind für Aussagen über die Leistungsfähigkeit einer Netzwerkkomponente unter realen Bedingungen meist nur eingeschränkt nutzbar. Durch die Verwendung fester Rahmenlängen entspricht die Belastung des DuT meist nicht dem Datenverkehr, der in realen Netzwerken anzutreffen ist. Die durchschnittliche Verteilung der Paketlängen in Internet Datenverkehr ist nach [JT2007] bimodal. Das heißt, Pakete der Länge 40 bis 100 Byte sowie die Längen 1400 bis 1500 Byte machen 44 % und 37 % alle Pakete aus. Dadurch ist anhand des RFC 2544 Durchsatz-Tests nur eine grobe Abschätzung der realen Leistungsfähigkeit durchführbar. Für die Bezeichnung von Datenströmen, die die Verteilung der Rahmenlängen des realen Internet Verkehrs nachbilden, wurde die Bezeichnung Internet Mix (IMIX) etabliert [MOR1999, S.2]. Es existierten jedoch keine Standards für die Zusammensetzung oder die Reihenfolge der Rahmenlängen in einem IMIX vgl. [HER2006]. Damit ist die Vergleichbarkeit der Messergebnisse, im Gegensatz zu denen der RFC 2544 Tests, nur bei Verwendung der selben Messsysteme und IMIX-Einstellungen gegeben.

2.4 Definition des IMIX Datenstroms

Tabelle 2: Beispiel IMIX Verteilung

Rahmenlänge L	**Anzahl N_s**	**Pakete / %**	**rel. Datenrate Dr_s / %**
64	4	40	4,765
520	4	40	38,719
1518	2	20	56,515

Da der Begriff IMIX nicht fest definiert ist, werden für diese Arbeit folgende Annahmen getroffen: Ein IMIX Datenstrom wird über eine Tabelle definiert, die jeder verwendeten Rahmenlänge eine Anzahl zuordnet. Die Reihenfolge, in der die Rahmen versendet werden, wird *Schedule* genannt und ist als Liste realisiert. Dieser wird

erzeugt, in dem aus jeder Rahmenlänge ein Paket entnommen und an den Schedule angehängt wird. Ist eine Länge leer, wird aus der nächsten Länge entnommen. Das geschieht so lange, bis alle Rahmenlängen aufgebraucht sind. In **Tabelle 2** ist ein Typischer IMIX definiert, **Tabelle 3** zeigt den daraus resultierenden Schedule.

Tabelle 3: Beispiel IMIX Schedule

Zeitpunkt	0	1	2	3	4	5	6	7	8	9
Rahmenlänge	64	520	1518	64	520	1518	64	520	64	520

2.5 Architektur der Intel Xeon E5-2600 v3 Plattform

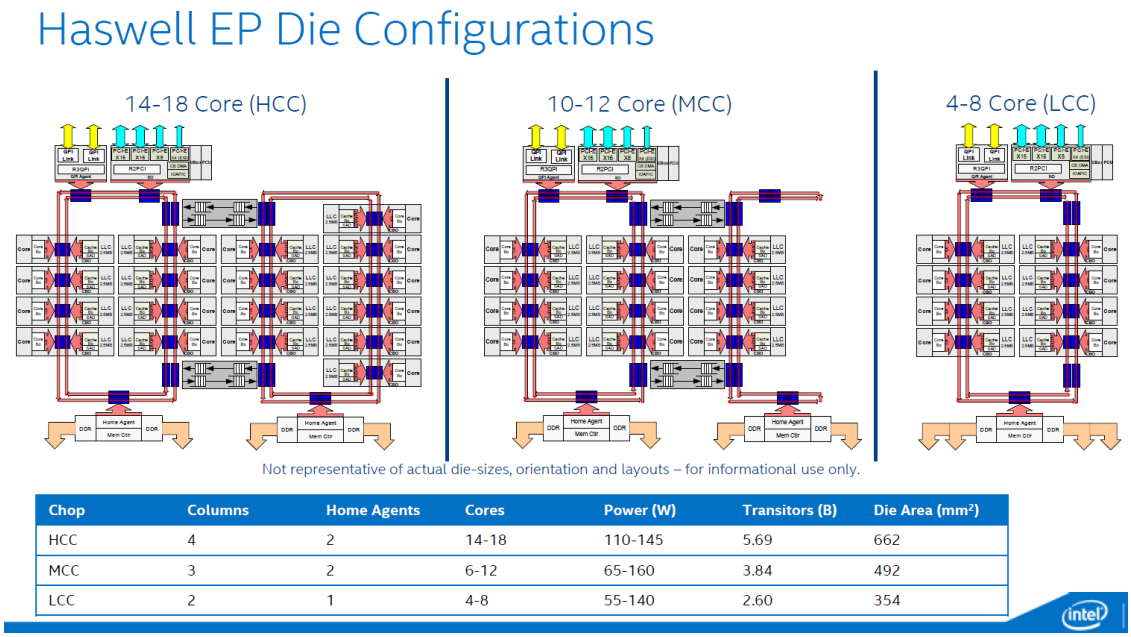
In diesem Abschnitt wird der Aufbau und die Besonderheiten einer aktuellen Serverplattform, am Beispiel der Intel Xeon 2600 v3 Reihe, erklärt. Ein tiefes Verständnis der Architektur ist nötig, um effiziente und performante Software zu entwickeln. Der Fokus liegt hier insbesondere auf der Anbindung von Peripherie und Hautspeicher, da diese für eine effiziente Paketverarbeitung von besonderer Bedeutung sind.

2.5.1 Aufbau der Plattform

Intel bietet mit der dritten Version der Xeon E5-2600er Reihe eine Plattform im mittleren Leistungsbereich an. Server Systeme mit bis zu zwei CPU Sockeln sind möglich. Für die Verbindung der beiden Sockel werden zwei Quick Path Interconnect (QPI) Schnittstellen eingesetzt. Die QPI Links werden mit 6,4, 8 oder 9,6 GT/s[4] betrieben, das entspricht einer Bruttodatenrate von bis zu 16 GByte/s je Richtung und Link. Einen klassischen Chipsatz, der Hauptspeicher, PCI-Express Bus und Prozessor verbindet, gibt es nicht mehr. Durch Integration der Arbeitsspeicher Schnittstelle und der PCI-Express Anbindung direkt in den Prozessor konnte der potenzielle Flaschenhals, welcher vorher durch die Anbindung des Chipsatzes vorhanden war, abgeschafft werden. Die Hauptspeicheranbindung hat 4 Speicherkanäle für Speichermodule nach DDR4 Standard und ermöglicht damit einen maximalen Speicherdurchsatz von 68 GByte/s.

[4] GigaTransfers pro Sekunde

Die einzelnen Komponenten der CPU werden intern durch einen oder zwei doppelt ausgeführte gegenläufige Ringbusse miteinander verbunden. Die CPUs der E5-2600 v3 Reihe sind in den drei Ausbaustufen LCC, MCC und HCC unterteilt. Der wesentliche Unterschied ist die Anzahl der CPU Kerne sowie deren Anordnung (siehe **Abbildung 4**). Es werden Prozessoren von sechs bis achtzehn Kernen angeboten. Dabei ist zu beachten, dass ein Modell mit sehr vielen Recheneinheiten eine geringere Taktfrequenz aufweist als ein Modell mit weniger Recheneinheiten. Der begrenzende Faktor ist hier die maximal mögliche Leistungsaufnahme und damit einhergehend die Abwärme der CPU.



Chop	Columns	Home Agents	Cores	Power (W)	Transitors (B)	Die Area (mm²)
HCC	4	2	14-18	110-145	5.69	662
MCC	3	2	6-12	65-160	3.84	492
LCC	2	1	4-8	55-140	2.60	354

Abbildung 4: Darstellung der Ausbaustufen der Intel Haswell Xeon CPUs (aus [KAR2013][11-13])

2.5.2 Besonderheiten der CPU

Durch die Anordnung der Rechenkerne (Cores) in zwei Cluster wird die CPU faktisch zu einem Non Uniform Memory Access (NUMA) System. Das bedeutet, dass ein Speicherzugriff, je nach Quelle und Ziel, unterschiedlich lang dauern kann. Das trifft auf alle MCC und HCC Typen, also jeder CPU mit mehr als acht Kernen, zu. Besonders Aufgaben, die einen hohen PCI-Express Durchsatz benötigen, sollten auf dem linken Cluster ausgeführt werden, da es näher am PCI-Express Interface ist.

Intel gibt eine Latenzreduktion von bis zu einem Prozent an [KAR2013, S.30], wenn die Software auf diese Besonderheiten Rücksicht nimmt.

Ein Zwei-Sockel-System ist, da der Speicher direkt an der CPU angebunden wird, auch eine NUMA Maschine. Da jede CPU eigene PCI-Express Verbindungen hat, ist es auch hier wichtig, diese Topologie in der Software zu beachten. Beispielsweise sollte möglichst vermieden werden, Speicherzugriffe eines PCI-Express Geräts an Sockel 0 auf Speicher an Sockel 1 durchzuführen, da der Weg länger und somit auch die Latenz höher ist. Bei NUMA Architekturen ist ein lokaler Speicherzugriff immer einem externen Zugriff vorzuziehen [MOL+2015].

Intel bietet auf dieser Plattform eine weitere Spezialfunktion an. Durch die Direct Cache Access (DCA) Technik ist es einem PCI-Express Gerät möglich, direkt den Cache der CPU für Lese- und Schreibzugriffe zu nutzen. Eine Anwendung hierfür ist der Empfangspfad von Netzwerkschnittstellen. Ohne DCA schreibt der Netzwerk-Chip ein empfangenes Paket zuerst in den Hauptspeicher, von dort werden die Daten, bei einem Zugriff der CPU, in die CPU geladen. Das hat zur Folge, dass für das Einlesen eines Datensatzes der Netzwerkschnittstelle zwei Hauptspeicherzugriffe nötig sind. Mit DCA kann diese Verzögerung erheblich verringert werden, da hier der Datensatz direkt in den CPU Cache geschrieben wird. Ein lokaler Speicherzugriff benötigt etwa 100 ns [MOL+2015]. Das entspricht bei einer Tankfrequenz von 2,6 GHz genau 260 Taktzyklen[5], ein Cache-Zugriff benötigt nur 12 Zyklen und ist damit etwa 21-mal schneller.

2.6 Funktionsweise moderner Ethernet Chips am Beispiel des Intel 82599EB

Ethernet Chips sind integrierte Schaltungen, die die Anbindung eines Computersystems an ein Ethernet Netzwerk realisieren. Durch immer höhere Daten- und Paketraten werden immer höhere Anforderungen gestellt. Um Ethernet Links mit 10 GBit/s und mehr bedienen zu können, werden mehr Funktionen aus dem Netzwerkstack in die Hardware verlagert. Moderne Netzwerk Chips bringen eine Vielzahl an Funktionen mit, um den Prozessor zu entlasten. Einige dieser sogenannten Offloading-Mechanismen sind für die Entwicklung eines Messsystems sehr nützlich. Der Intel 82599EB Chipsatz bietet zwei 10 GBit/s Ethernet Schnittstellen und wird über PCI-Express 2.0 x8 angebunden. In den folgenden Abschnitten wird das Funktions-

[5] $(260 Zyklen = 2,6 GHz * 100 ns)$

prinzip der Intel 82599EB Netzwerk Chips erklärt und auf die für ein Messsystem hilfreichen Offloading Funktionen eingegangen.

2.6.1 Aufbau der Funktionseinheiten

Die beiden Schnittstellen erscheinen dem System gegenüber als dedizierte PCI Geräte. Deren Architektur ist in eine Kontroll- und Datenebene unterteilt. Die Datenebene besteht aus dem Empfangs- und Sendepfad, welche sich nur die PCI Express Schnittstelle und den Media Access Controller (MAC) teilen und ansonsten unabhängig voneinander sind. Über die Kontrollebene wird der Chip konfiguriert und gesteuert. Durch das geringe Datenaufkommen wird hier mit Memory Mapped Input Output (MMIO) gearbeitet. Für den Datenpfad wird Direct Memory Access (DMA) eingesetzt, um die erforderliche Datenrate zu bewältigen, ohne den Hauptprozessor damit unnötig zu belasten. DMA ermöglicht es dem Netzwerkchip Daten selbständig mit Hauptspeicher auszutauschen. Um die Parallelität moderner Prozessoren ausnutzen zu können, werden für das Senden und Empfangen der Netzwerkrahmen bis zu 128 unabhängige Warteschlangen (Queues) pro Richtung unterstützt. Jede dieser Queues wird durch einen Ringpuffer im Hauptspeicher des Rechners realisiert, in welchem die Metadaten der Rahmen gespeichert werden. Dazu gehören unter andrem die Adresse des Paketpuffers, in dem der Rahmen gespeichert ist und die Länge des Rahmens.

RX Pfad

Der Empfangspfad bietet eine ganze Reihe von hardwarebasierten Filtermechanismen, die es ermöglichen, empfangene Rahmen nach diversen Kriterien zu verwerfen oder auf die Warteschlangen zu verteilen. Neben den Filtern können auch die Prüfsummen vom Ethernet, IP, TCP sowie UDP überprüft werden. Von besonderem Interesse sind hier die sogenannten 5-Tupel Filter. Mit diesen Filtern ist es möglich, auf das Protokoll Feld im IP-Header, die Quell- und Zieladresse sowie den Quell- und Zielport eines Rahmens als Filterkriterium zu benutzen, um ihn einer bestimmten RX-Queue zuzuordnen. Jedes der 5 Filterkriterien lässt sich über eine Bitmaske ein- oder ausschalten, womit auch weniger spezifische Filter möglich sind. Es können bis zu 128 dieser Filter konfiguriert werden.

Mit der *Flow Director* genannten Technologie bietet Intel eine weitere Möglichkeit Pakete noch umfangreicher zu filtern. Der Unterschied zum 5-Tupel Filter besteht im wesentlichen darin, dass Flow Director Filter für die einzelne Felder Masken anbieten und es möglich ist, auf ein 2 Byte Tupel innerhalb der ersten 64 Byte des Rahmens, zu filtern. Die Masken bieten die Möglichkeit nur auf bestimmte Bits eines Feldes zu filtern und damit beispielsweise statt einer IP Adresse ein ganzes Subnetz als Kriterium zu verwenden. Diese Masken sind jedoch Global gültig, das heißt, sie wirken sich auf alle Flow Director Filter aus. Nach dem ein empfangener Rahmen die Filter passiert hat und einer RX Queue zugeordnet wurde, wird ein Paket Deskriptor in die entsprechende Queue eingereiht.

TX Pfad

Um das Senden von Paketen zu unterstützen, bietet der Intel 82599EB Chip diverse Offloading Funktionen. Für Ethernet, IPv4, UDP sowie TCP Pakete können alle Prüfsummen in Hardware berechnet werden, was die Host CPU erheblich entlastet. In diesem Kontext wird auch das Segmentieren[6] von TCP und UDP Paketen als Hardwarefunktion zur Verfügung gestellt.

Eine weitere wichtige Funktion ist der Transmit Rate Scheduler (TRS): Jede der 128 TX Queues kann individuell in der Datenrate beschränkt werden. Durch einen programmierbaren Rate Factor (RF) und die Paketlänge PL des zuletzt gesendeten Pakets wird der Minimal Inter Frame Space ($MIFS$) nach der **Formel** (6) berechnet. Die Paketlänge PL bezeichnet hier die Länge des Ethernet Rahmens in Byte. Der Rate Factor RF wird mittels Division der Linkdatenrate D_{link} durch die Zieldatenrate D_{target} berechnet. Daraus ergibt sich Byte als Einheit für $MIFS$. Für 64 Byte Rahmenlänge ergibt sich die maximale Schrittweite durch **Gleichung 7** ($frameSize = 64\ Byte;\ n = MIFS = 1\ Byte$) zu 1,118 % der maximalen Ethernet Datenrate für den erste Schritt, L_{over} ist wie in Abschnitt 2.2 beschrieben 20 Byte die sich aus 12 Byte IFG und 8 Byte Präambel zusammensetzen. Mit steigender Rahmenlänge als auch Minimun Inter Frame Spacing (MIFS) sinkt die Schrittweite, dadurch steigt die erzielbare Auflösung der einstellbaren Datenrate. In **Abbildung 5** sind die Schrittweiten des TRS über der prozentuale Linkrate für einige Rahmenlängen dargestellt.

[6] Beschreibt das Aufteilen großer Datenpuffer in mehrere kleinere Netzwerkpakete, ebenso das wieder Zusammensetzen im RX Pfad

$$MIFS = PL * RF = PL * \frac{D_{link}}{D_{target}} \tag{6}$$

$$loadStep(n, frameSize) = \frac{frameSize + L_{over}}{frameSize + L_{over} + (n-1)} - \frac{frameSize + L_{over}}{frameSize + L_{over} + n} \tag{7}$$

TRS Auflösung in Abhängigkeit der relativen Datenrate und Rahmenlänge

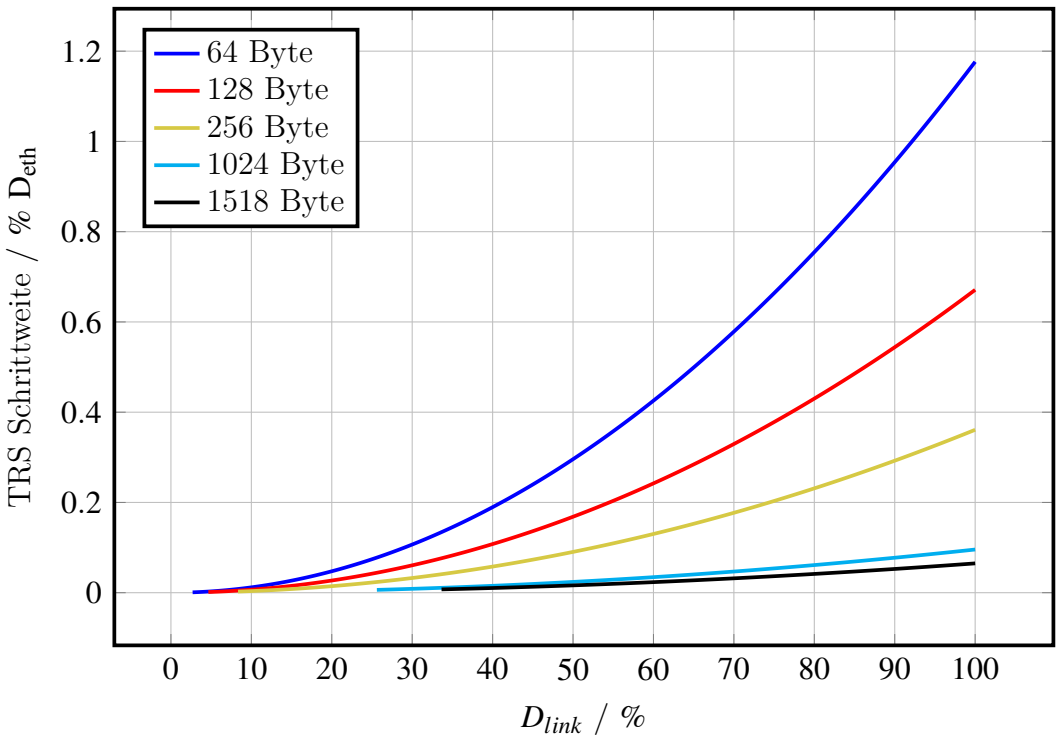


Abbildung 5: TRS Auflösung in Abhängigkeit der relativen Datenrate und Rahmenlänge

2.6.2 Hardwarezeitstempel

Für die Zeitsynchronisation von Computersystemen wurde das Precision Time Protocol (PTP) entwickelt und in dem IEEE1588 Standard spezifiziert. Um die Implementierung einer sehr genauen Synchronisation zu ermöglichen, hat Intel Zeitstempel Mechanismen für bestimmte Pakete im Chipdesign integriert. Es können für das

Senden und Empfangen je ein Zeitstempel in einem Register des Chips gespeichert werden. Die Auflösung dieser Zeitstempel hängt von der Linkdatenrate D_{link} ab und ist in **Tabelle 4** zu finden.

Tabelle 4: Intel 82599EB: Timer Granularität

D_{link}	Zähler Frequenz	Granularität
10 GBit/s	156,25 MHz	12,8 *ns*
1 GBit/s	15,625 MHz	128 *ns*
100 MBit/s	1,5625 MHz	1,280 μs

2.7 Paketverarbeitung mit spezialisierter Software

In dem Paper [RIZ2012] kommt Rizzo zu dem Schluss, dass vor allem durch das Kopieren der Daten, Metadatenverwaltung und Aufwand für Systemrufe, die Leistungsfähigkeit der Paketverarbeitung in Computersystemen maßgeblich beeinflusst wird. Um die Probleme, die bei der Verarbeitung hoher Paketraten in konventionellen Netzwerkschichten auftreten, zu umgehen, wurden neue Ansätze entwickelt. Zwei dieser Ansätze werden in diesem Abschnitt näher erklärt. Das von Luigi Rizzo im dem oben genannten Paper vorgestellte *netmap* API als auch das von Intel vorangetrieben DPDK verfolgen ein ähnliches Konzept: Die Netzwerkpakete werden nicht mehr von einem im Betriebssystemkern laufendem Netzwerkstack, sondern direkt in einem Userspace Prozess verarbeitet. Außerdem wird ausschließlich Polling angewendet um Software und Hardware zu synchronisieren. Beide Ansätze verfolgen durchgehend die Strategie, Pakete nicht einzeln, sondern in Gruppen, auch Bulk genannt, zu verarbeiten. Dadurch lässt sich der Aufwand, der für das Verwalten der Hardware Ringpuffer anfällt, auf mehre Pakete verteilen. Ein weiteres grundlegendes Konzept wird „zero copy“ genannt: Die eigentlichen Daten werden nie kopiert, es werden ausschließlich Referenzen verwendet.

2.7.1 Das netmap API

Netmap wird hauptsächlich für FreeBSD entwickelt und ist seit Version 9 regulär enthalten. Es existiert auch ein Portierung für den Linux Kernel [7]. Netmap ist eine Kernel API und bietet die Möglichkeit, am Netzwerkstack vorbei, die RX und TX

[7] https://github.com/luigirizzo/netmap/LINUX

Queues einer Netzwerk Schnittstelle zu bedienen. Dazu wird für jede konfigurierte Queue ein Ringpuffer angelegt. Diese Ringpuffer werden per *mmap()* Systemruf in den Prozess Adressraum eingebunden. Sie ermöglichen so das Senden und Empfangen, ohne dass die Daten zwischen Kernel und Userland kopiert werden müssen. Da der Speicher für die Ring Puffer und Paket Puffer vollständig im Kernel alloziert wird, muss zur Laufzeit der Anwendung kein dynamischer Speicher alloziert werden. Die Synchronisation erfolgt per *poll()* oder *select()* Systemruf. Die Ring Puffer Zeiger werden dabei mit denen des Netzwerkchips abgeglichen. Da die Netzwerkkarte direkt auf den Speicher zugreifen kann, ist auch hier kein Kopieren der Daten nötig.

Der Treiber für den Netzwerk Chip bleibt, im Gegensatz zu DPDK, im Kernel, es wird lediglich der Datenpfad umgeleitet. Dadurch ist es möglich, dass die Netzwerkschnittelle auch weiterhin mit Standard Systemprogrammen verwaltet werden kann. Außerdem wird so verhindert, dass der User-Prozesse direkten Hardwarezugriff hat und damit das gesamte System zum Absturz bringen kann. Da das Netmap API generisch ist, finden sich keine Möglichkeiten, um spezielle Offloading Funktionen der Netzwerkchips zu verwenden.

2.7.2 Data Plane Development Kit (DPDK)

Das DPDK ist ein Applikations-Framework für die Entwicklung hoch performanter Paketverarbeitungssysteme. Es wird als Open Source Projekt mit BSD Lizenz entwickelt und hauptsächlich von Intel und Windriver vorangetrieben. Durch das offene Entwicklungsmodell gibt es mittlerweile auch Unterstützung für Netzwerkchips von mit Intel konkurrierenden Unternehmen. Die Hauptplattform sind Intel X86 basierte Rechner mit einem Linux Betriebssystem. Das DPDK besteht im wesentlichen aus drei Komponenten: Dem Environment Abstraction Layer (EAL), diverse Poll Mode Drivers (PMDs) für die Netzwerkschnittstellen und einer ganzen Reihe an Bibliotheken. Alle Komponenten laufen innerhalb eines Prozess-Kontextes im Userspace, inklusive der Treiber für die Netzwerkschnittstellen.

Der EAL kapselt die Eigenheiten verschiedener Umgebungen und bietet der Applikation eine konsistente Schnittstelle. Unter einer Umgebung wird hier die Kombination aus Architektur (Intel/ARM, 32/64 Bit) des verwendeten Compilers sowie des Betriebssystems verstanden. Um dies zu gewährleisten, werden im EAL unter anderem Funktionen zur Speicherverwaltung, Thread Synchronisation, PCI Bus Zugriff

und CPU-Affinität[8] gekapselt. Das Sinn ist, dass eine Anwendung dadurch von der Umgebung unabhängig ist und somit auf jeder von DPDK unterstützen Plattform lauffähig ist.

Die PMDs bilden die Schnittstelle zur Hardware, sie kapseln den Treiber für die Netzwerk Chips. Da der Treiber somit im Kontext der Applikation ausgeführt wird, sind keine Systemrufe für die Kommunikation mit der Netzwerkhardware notwendig. Das ist möglich, da über einen auf Linux UIO basierenden Treiber die PCI Speicherbereiche der Hardware aus dem Userspace zugänglich sind. Dieser Treiber ist die einzige DPDK Komponente die im Kernel ausgeführt wird.

Für eine DPDK Anwendung ist es, abgesehen von Chip-spezifischen Offloading Funktionen, nicht relevant, welcher Netzwerkchip verwendet wird. Für die Grundfunktionen steht ein einheitliches API zur Verfügung. Für spezielle Funktionen der Chips wie Prüfsummenberechnung oder Segmentierung existiert jedoch kein PMD übergreifendes API.

Die drei wichtigsten Bibliotheken sind *librte_ring*, *librte_mempool* und *librte_mbuf*. Sie stellen alle Grundfunktionalitäten für die Verarbeitung von Paketen zur Verfügung. In DPDK werden Ring Puffer mit fester Größe eingesetzt, um Komponenten der Verarbeitungspipeline zu verbinden. Diese werden von *librte_ring* implementiert. Ein zentrales Konzept in DPDK sind die Memory Polls, in denen Speicherblöcke fester Größe verwaltet werden. Die Implementierung in *librte_mempool* sorgt dafür, dass die Blöcke gleichmäßig auf alle Speicherkanäle einer CPU verteilt werden. Außerdem wird sicher gestellt, dass ein Block nicht mehrere Kanäle überlappt. Es ist auch möglich vorzugeben, auf welchem CPU Sockel der Speicher, falls ein NUMA System verwendet wird, alloziert werden soll. Damit wird gewährleistet, dass eine Netzwerkkarte, die an Sockel 1 angeschlossen ist, nicht auf Speicher, der an Sockel 0 betrieben wird, schreiben muss. Bei der Rückgabe von Speicherblöcken an den Pool wird die CPU-Nummer (lcore_id) des Threads erfasst. Dadurch kann das Speicherobjekt in einem Cache vorgehalten werden der über die CPU Nummer indiziert wird (**Abbildung 6**). Der Sinn dahinter ist, dass bei der nächsten Anfrage nach Speicherblöcken zuerst jene ausgeliefert werden, die schon einmal auf dem anfragenden CPU Kern genutzt wurden. Da die Caches exklusiv von einer CPU benutzt werden, ist zudem weniger Locking nötig um Operationen auf dem Mempool auszuführen.

Durch *librte_mbuf* wird eine Datenstruktur bereit gestellt, mit der Netzwerkpakete dargestellt werden können. Ein *mbuf* besteht aus einem zusammenhängenden Speicherbereich, der in Metadaten und einen Datenpuffer konstanter Länge aufgeteilt ist.

[8] Zuordnung eines Threads zu einem CPU Kern

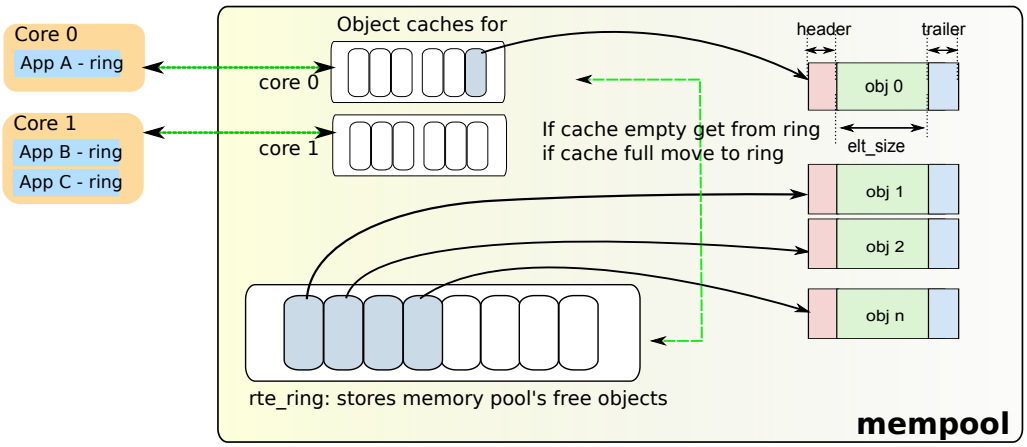


Abbildung 6: DPDK Mempool Cache

Die Metadaten beinhalten unter anderem die Länge und das Offset des Datenpuffers sowie einen Zeiger auf einen weiteren *mbuf*. Durch diesen Zeiger ist es möglich, Pakete die länger als ein *mbuf* sind, durch Verkettung darzustellen.

Eine DPDK Applikation sollte Pakete möglichst in Blöcken verarbeiten, um den Aufwand, der pro Hardwarezugriff entsteht, auf mehrere Pakete zu verteilen.

3 Entwurf und Architektur des Messsystems

Im diesem Kapitel wird der Entwurf für ein Messsystem vorgestellt, dass Netzwerkomponenten der OSI-Schicht 3 mit Datenraten bis 10 GBit/s vermessen kann. Das Messsystem soll alle Tests des RFC2544 sowie einen IMIX basierten Durchsatz-Test durchführen können. Die Messgrößen werden nach Abschluss aller Test in Form eines Messprotokoll ausgegeben. In den folgenden Abschnitten werden die Anforderungen, die daraus resultierende Auswahl der Hardware und Software, sowie die Architektur des Messsystems erklärt. Das entwickelte Messsystem wird im Folgenden als **nettest** bezeichnet.

3.1 Anforderungen

Neben den im Abschnitt 2.3.3 aufgelisteten Anforderungen für Tests nach RFC2544 werden in diesem Abschnitt weitere Anforderungen an dass Messsystem erarbeitet.

Der hauptsächliche Einsatzbereich des Messsystems ist die entwicklungsbegleitende Vermessung von VPN Gateways. Für die Auswahl der Hardware Komponenten neuer Systeme ist es erforderlich, deren Leistungsfähigkeit zu ermitteln. Da nicht ausschließlich 10 GBit Ethernet verwendet wird ist es nötig, dass das Messsystem auch mit 1 Gigabit Ethernet verwendbar ist. Konkret werden die Ausprägungen 1000BASE-T, 1000BASE-SX, 10GBASE-SR sowie 10GBASE-T benötigt.

Außerdem soll als Medium neben Glasfaser (1000BASE-SX und 10GBASE-SR) auch Kupfer (1000BASE-T und 10GBASE-T) unterstützt werden. Für die Softwareentwicklung der VPN-Gateway ist es sinnvoll, dass die Messungen automatisierbar sind, um automatische Test zu ermöglichen. Dies erfordert, dass die Ergebnisse und Testparameter maschinenlesbar abgelegt werden, um die automatische Auswertung und spätere Nachvollziehbarkeit der Ergebnisse zu gewährleisten.

Neben den Tests, die im RFC2544 beschrieben sind, soll auch ein Durchsatztest mit einem IMIX Datenstrom ermöglicht werden. Der Test wird an das Verfahren des RFC2544 Durchsatztests angelehnt.

3.2 Auswahl der Software und Hardware

In diesem Abschnitt wird MoonGen, ein Paketverarbeitungs-Framework basierend auf DPDK, vorgestellt. Anschließend wird der Implementierungsaufwand, gegenüber einer direkt auf DPDK aufbauenden Lösung, abgeschätzt. Darauf folgend wird die Wahl der Hardware-Plattform sowie die verwendeten Komponenten beschrieben.

3.2.1 MoonGen

MoonGen ist eine, auf DPDK und *LuaJIT* aufbauende Software, die entwickelt wurde um hochperformante, flexible Paketgeneratoren mit Standard Hardware realisieren zu können. Es wurden damit schon Paketraten von 178,5 Mpps bei einer Datenrate von 120 GBit/s erreicht (sieht [EMM+2015]). MoonGen bietet neben dem Erzeugen von Paketströmen auch die Möglichkeit, empfangene Pakete sehr performant zu verarbeiten und damit die Grundfunktionen für leistungsfähige Paketverarbeitungssysteme.

Lua ist eine effiziente, schlanke Skriptsprache, die sich einfach in andere Anwendungen einbetten lässt. Mit LuaJIT steht ein Laufzeitsystem für Lua zur Verfügung, dass besonders auf Geschwindigkeit und die Integration in andere Anwendungen optimiert ist. Der Lua Code wird zur Laufzeit, durch einen Just-In-Time Compiler, in Maschinencode übersetzt. Dadurch ist der, bei interpretierten Scriptsprachen übliche Nachteil der geringen Ausführungsgeschwindigkeit gegenüber kompilierten Sprachen, vernachlässigbar. LuaJIT bietet mit der FFI Bibliothek eine Schnittstelle, um Funktionen und Datenstrukturen von C Bibliotheken direkt in Lua Applikationen zu verwenden und ermöglicht es damit, Code-Abschnitte, die besonderer Optimierung bedürfen, in C zu programmieren.

MoonGen initialisiert die DPDK Bibliothek und stellt einige performancerelevante Subroutinen als C Implementierung bereit. Die eigentliche Applikationslogik wird durch das auszuführende Lua-Skript realisiert. Nachdem DPDK durch MoonGen initialisiert ist, wird das Skript durch eine LuaJIT Instanz abgearbeitet. Weil Lua

keine parallele Code Ausführung vorsieht, wird für jeden weiteren Thread, der aus dem *master* Thread heraus gestartet wird, eine, in eine DPDK Thread gekapselte LuaJIT Instanz, gestartet. Da Lua keine Nebenläufigkeit vorsieht, existieren in den Lua Kontexten keine gemeinsamen Variablen. MoonGen zeichnet sich durch eine umfangreiche Unterstützung für die gängigen Protokoll Header Strukturen, einschließlich der Berechnung etwaiger Prüfsummen, aus. Außerdem existiert eine sehr genaue Methode zur Laufzeitmessung auf Basis der PTP Funktionalität der Intel 82599 Netzwerk Chips. Für die Messung der Daten- und Paketraten wird auf Statistikzähler der Netzwerkschnittelle zurückgegriffen.

Schwächen der MonnGen RFC2544 Testsuite

Mit MoonGen werden diverse Beispiel-Skripte ausgeliefert, die auch eine Implementierung des RFC2544-Verfahrens beinhalten, diese hat jedoch diverse Schwächen: Alle Tests werden nur unidirektional ausgeführt, es ist jedoch üblich, dass bidirektional getestet wird. Der RFC2544 fordert nicht explizit, dass gleichzeitig in beide Richtungen gesendet wird, es wird jedoch in [BM1999, Abschnitt 14] empfohlen, um realistischerer Ergebnisse zu bekommen. Da in realen Netzen der Datenverkehr so gut wie nie nur in eine Richtung fließt, ist es unbedingt nötig, die Tests bidirektional durchzuführen, um aussagekräftige Daten zur Leistungsfähigkeit zu erhalten. Auch der Latenztest entspricht nicht dem RFC2544 Vorgaben: Das Verfahren zur Latenzmessung, erzeugt einen Datenstrom mit einer gewissen Datenrate $D_{latency}$, die zusätzlich zum Lastdatenstrom D_{laod} auf das DuT wirkt. Damit wird die Latenz nicht wie gefordert bei D_{load}, sondern bei $D_{load} + D_{latency}$ gemessen. Dadurch ergeben sich unter Umständen Paketverluste, da die Gesamtdatenrate über dem ermittelten Maximum liegt. Weil der implementierte Latenztest nicht auswertet, ob alle Pakete des Lastdatenstroms empfangen werden, fällt das Problem nicht direkt auf. Die Ergebnisse des Latenztest sind aufgrund der höheren Datenrate nicht Standard konform. Der RFC2544-Standard empfiehlt die Verwendung einer Sequenznummer in jedem Lastpaket, die während des Empfangs kontrolliert werden soll. Die Implementierung der Tests im MoonGen RFC2544 biete diese Funktion nicht. Grundsätzlich besteht ein Problem bei der Verwendung des MoonGen *memory* Moduls: Ein *Memory Pool* kann, bedingt durch die DPDK Architektur, nicht wieder freigegeben werden. Dadurch ist die Anzahl der Pool Instanzen durch den verfügbaren Hauptspeicher begrenzt. In jedem der Implementierten Test werden für jeden Testschritt, zwischen 1 und 4 neue *Memory Pool* Objekte angelegt. Allein für den Durchsatztest, mit einer Granularität von 1 MBit/s bei einer Linkrate von 10 GBit/s, ergibt sich

eine Anzahl von mindestens 224[1] Speicher Pool Objekten. Das entspricht, unter der Voraussetzung dass die Standard Parameter verwendet werden, mindesten 896 MiB[2] Speicher, der nicht wiederverwendet werden kann. Dieses Speicherleck führt früher oder später zum Absturz der Anwendung.

Mit MoonGen existiert eine gute Basis zur Entwicklung eines Messsystems, basierend auf handelsüblicher Standard Hardware. Durch die Kombination von DPDK und Lua ist es sehr schnell und komfortabel möglich, hoch performante Paketverarbeitungssysteme zu entwickeln. Was MoonGen besonders attraktiv macht, ist der große Funktionsumfang der mitgelieferten Bibliotheken. Diese reduzieren den Aufwand gegenüber einem rein DPDK basierten System erheblich. Es ist aufgrund der im vorherigen Absatz genannten Unzulänglichkeiten nicht sinnvoll, diese RFC2544 Implementierung als Basis zu verwenden. Die nötigen Änderungen wären sehr umfangreich, da strukturelle Anpassungen erforderlich sind. Im Folgenden Kapitel wird daher der Entwurf eines Messsystems vorgestellt, der diese Schwächen nicht aufweist.

3.2.2 Aufwand der Implementierung

Das Messsystem wird auf MoonGen basieren da es gegenüber DPDK, wie in vorherigen Abschnitt dargestellt, viele der benötigten Komponenten bereits mitbringt. Insbesondere das Verfahren zur Laufzeitmessung sowie die Protokoll Implementierungen müssen für eine, direkt auf DPDK aufsetzende Lösung, neu geschrieben werden. Weiterhin spricht für MoonGen als Basis, dass große Teile der Applikation in Lua geschrieben werden können. In **Anhang A - Tabelle 18** ist eine genaue Gegenüberstellung der Funktionalitäten aufgelistet.

3.2.3 Auswahl der Hardware

Um alle geforderten Ethernet-Standards zu ermöglichen, sind vier Schnittstellen nötig, zwei mal Kupfer und zwei mal Small Form-factor Pluggable Plus (SFP+), für die optischen Verbindungen. Für die Entwicklung stehen zwei Karten vom Typ Intel X520-SR1 sowie eine Intel X520-T2 zur Verfügung. Diese werden vom MoonGen unterstützt. Die X520-SR1 verfügt über einen SFP+ Steckplatz und bietet, ausge-

[1] 2 Pool (RX/TX) * 8 Rahmenlängen * 14 Suchschritte = 224 Pools
[2] 2048 Byte * 2047 Blöck * 224 Pools = 896 MiB

stattet mit einem Optic Modul des Typs E10GSFPSR, Konnektivität für optisches Ethernet nach 10GBASE-SR sowie 1000BASE-SX Standard (siehe [INT2012]). Die Anbindung der zwei kupferbasierten Schnittstellen nach Standard 1000BASE-T und 10GBASE-T wird durch die X520-T2 Karte realisiert. Beide Kartentypen sind mit einem Intel 82599 Chip ausgestattet und werden über die PCI-Express v2.0 8-fach Schnittstelle mit dem System verbunden. Somit muss das Computersystem über drei Steckplätze verfügen, die mindestens diesem Standard entsprechen. Eine höhere Spezifikation ist bis zur aktuellen Version 4 unproblematisch, da PCI-Express Abwärtskompatibilität vorsieht (siehe [PCI2016]).

Als Hardware-Plattform für das Messsystem wird ein Intel Xeon System bevorzugt, da DPDK vor allem durch Intel vorangetrieben wird und daher besonders für deren Systeme optimiert ist. Um genügend Leistungsreserven zu haben wurde ein Dell PowerEdge R530 Server mit zwei Intel Xeon 2660v3 CPU beschafft. Der R530 wurde ausgewählt, weil die PCI-Express Steckplätze nicht ausschließlich an einem CPU Sockel hängen. Dadurch ist es möglich, das eine CPU nur eine Netzwerk Schnittstelle bedient. Die zweite CPU muss deshalb nicht den Umweg über den QPI Bus nehmen, was diesen als möglichen Flaschenhals ausschließt. Um ein ausgewogenes Preis-Leistung-Verhältnis zu gewährleisten, muss bei der Auswahl der CPU ein Kompromiss zwischen Taktfrequenz und der Anzahl der Kerne gefunden werden. Der Xeon 2660v3 Prozessor mit 10 Kernen und einem Basistakt von 2.6GHz sollte es ermöglichen, dass zwei Schnittstellen an einem Sockel betrieben werden können, ohne dass Threads auf dem anderen Sockel ausgeführt werden müssen. Um eine möglichst hohe Speicherbandbreite zu erzielen, sind pro Sockel drei DDR4- Speichermodule mit je 8 GiB Speicher verbaut. Jedes Modul wird von einem eigenen Speicherinterface bedient, damit wird pro Sockel eine Datenrate von 51,2 GB/s erreicht. In **Tabelle 5** sind die verbauten Komponenten zusammengefasst, **Abbildung 7** zeigt den Aufbau des Systems. Der Preis für das System inklusive der Netzwerkkarten liegt bei etwa 6500 €.

Tabelle 5: Hardware des Entwicklungssystems

Modell	Dell Poweredge R530
CPU	2x Intel Xeon 2660v4
RAM	6x 8GiB DDR4-2133
NIC	2x Intel X520-SR1 + 1 Intel X520-T2

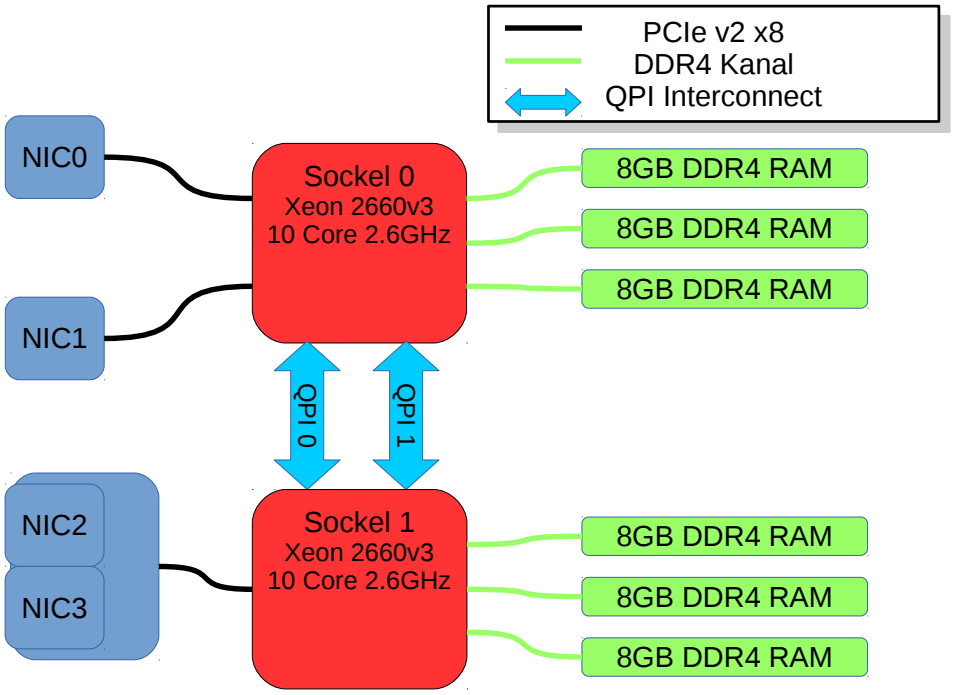


Abbildung 7: Systemarchitektur Dell PowerEdge R530

3.3 Software Architektur

Die Software besteht aus mehreren Threads, um die einzelnen Aufgaben auf dedizierte CPU Kerne zu verteilen. In Abbildung 8 ist der gesamte Aufbau dargestellt: Nachdem der **Master-Thread** die Konfiguration eingelesen hat steuert er den gesamten Testablauf und erzeugt schlussendlich den Testreport. In dem Kasten **Paketverarbeitung** sind alle Aufgaben zusammengefasst, welche über die Netzwerkschnittellen kommunizieren müssen. Der **ARP-Thread** läuft im Gegensatz zu den **RX-,TX- und Laufzeitmessungs-Threads** immer. Das ist erforderlich, um zu jeder Zeit auf ARP-Requests des DuT antworten zu können. Die anderen Threads der Paketverarbeitung werden für jeden Trail mit anderen Parametern neu gestartet.

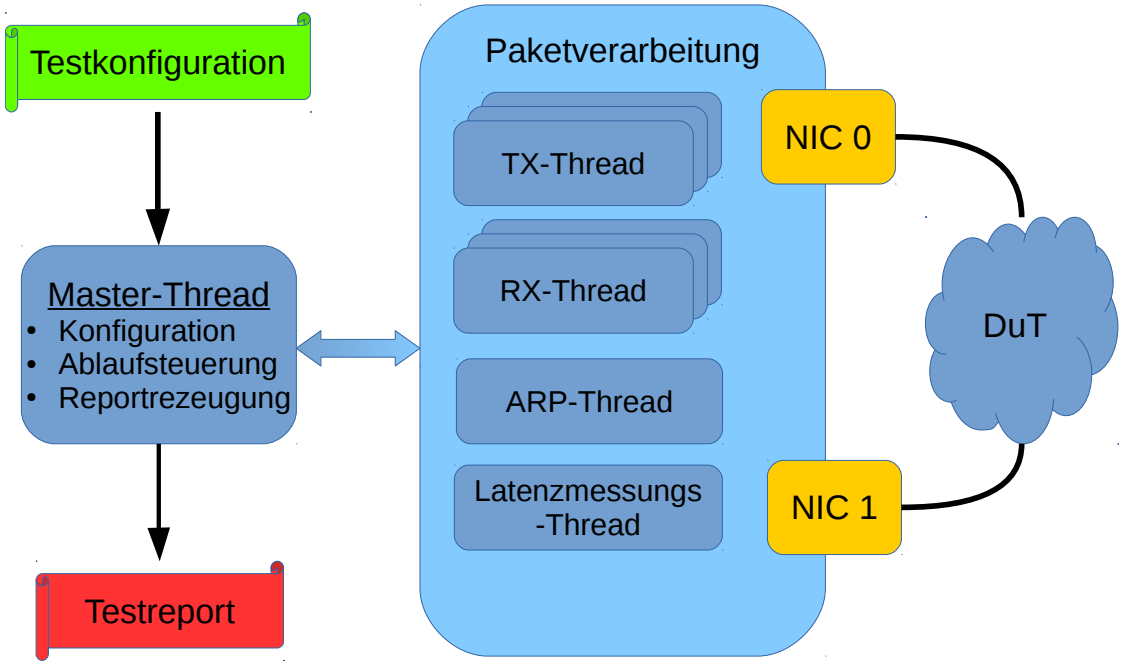


Abbildung 8: Softwarearchitektur des Messsystem

3.3.1 Das Stream Konzept

Um die Eigenschaften der generierten Lastpakete eines Datenstrom zu beschreiben, wird ein **Stream** definiert. In dieser Datenstruktur werden alle Parameter, wie die Adressen der Ethernet und IP Header, UDP Portnummern sowie die Datenrate und die Länge des Rahmens gespeichert. Zusätzlich muss das Quell- und Ziel-Interface gespeichert werden. Der zweite Stream für bidirektionale Tests lässt sich trivial aus dem ersten ableiten, daher kann ein Stream als bidirektional markiert werden. Die Software erzeugt dann die zweite Definition für die Gegenrichtung automatisch. Somit kann Redundanz in der Konfiguration vermieden werden.

3.3.2 Datenpfad

Für die Tests nach RFC2544 sowie den IMIX-Test ist es erforderlich, dass gleichzeitig zwei Paketströme durch den Tester generiert werden. Außerdem müssen die, durch das DuT weitergeleiteten Pakete, ausgewertet werden. Die Laufzeitmessung erfolgt parallel dazu als separater Datenstrom. Dadurch ist es nicht nötig, in jedem Paket zusätzlich zur Sequenznummer einen Zeitstempel zu speichern. Um die sehr hohe Paketrate von insgesamt 59,52 Mpps[3] zu bewältigen, muss eine parallele Paketverarbeitung entworfen werden. In **Abbildung 9** ist der Aufbau des Datenpfads dargestellt.

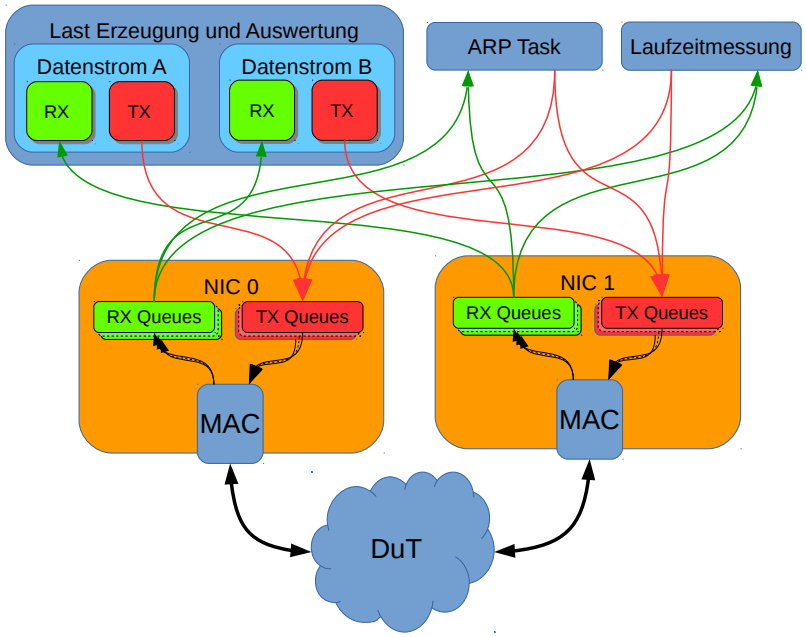


Abbildung 9: Datenpfad Schema

Der Datenpfad besteht aus drei Komponenten, die Zugriff auf die Netzwerkschnittstellen benötigen: In der Komponente **Last Erzeugung und Auswertung** werden die Testpakete im, mit TX bezeichneten TX-Thread, erzeugt und im RX-Thread ausgewertet. Dazu wird jedem Datenstrom eine TX Queue der sendenden Schnittstelle und eine RX Queue der empfangenden Schnittstelle zugeordnet. Die Begrenzung der Sendedatenrate kann somit über den TRS der TX Queue pro Stream erfolgen.

[3] 59,52 Mpps = 2 x 14,88Mpps Tx + 2 x 14.88Mpps RX , 14,88Mpps ist die maximale Paketrate für 10GBit Ethernet

Die Komponenten **ARP-Task** und **Laufzeitmessung** benötigen pro konfiguriertem Netzwerkinterface je eine RX und TX Queue. Um die Datenströme, die auf einer Schnittstelle empfangen werden, der richtigen RX Queue zuzuordnen, wird je Strom ein 5-Tupel Filter (siehe Abschnitt 2.6.1) der Netzwerkkarte eingesetzt. Das erfordert jedoch, dass sich die Datenströme mindestens in einem der fünf Felder des IP Headers unterscheiden.

Da die eingesetzten Netzwerkchips 128 Queues zur Verfügung stellen, können mit dieser Architektur theoretisch 126 individuelle Datenströme modelliert werden. Praktisch ist die Anzahl der parallelen Datenströme jedoch durch die verfügbaren CPUs begrenzt. Für das Erreichen hoher Paketraten ist es erforderlich, dass je Datenstrom zwei CPU Kerne verwendet werden. Eine CPU übernimmt das Erzeugen und Senden, die andere empfängt und wertet die Sequenznummern aus. Damit ergibt sich für das gesamte Messsystem der Bedarf an CPUs: $N_{CPU} \geq 2 * N_{Stream} + 3 + 1$. Die Drei steht für Master- ,ARP- und Laufzeitmessungsthread. Eine weitere CPU muss für das Betriebssystem eingeplant werden. Somit werden voraussichtlich 8 CPUs benötigt

3.3.3 Auswertung der Sequenznummern

Eine wichtige Funktion des Messsystems ist, die Reihenfolge sowie fehlende und doppelte Pakete zu erkennen. Dazu ist es erforderlich, dass jedes gesendete Lastpaket eindeutig identifiziert werden kann. Zu diesem Zweck wird in den ersten 64 Bit der UDP Nutzdaten eine monoton wachsende Zahl, die Sequenznummer, übertragen. Somit können 2^{64} Pakete eindeutig unterschieden werden. Die Sequenznummer startet bei 0 und wird für jedes weitere Paket inkrementiert. Die Reihenfolge, in der die Pakete durch die Netzwerkkarte versendet werden, ist nur deterministisch, wenn pro Sequenznummerendomäne eine TX-Queue verwendete wird. In Verbindung mit dem TRS ist die Reihenfolge, mit der Pakete aus den einzelnen Queues entnommen und versendet werden, nicht mehr vorhersehbar. Die Auswertung der Sequenznummer ist gegenüber dem Erzeugen um einiges aufwändiger. Der naive Ansatz wäre, dass der Empfänger für jede Sequenznummerndomäne die zuletzt empfangene Nummer mit der des aktuellen Paketes vergleicht. Es wird erwartet, dass der aktuelle Wert um eins größer ist als der vorherige. Damit lässt sich jedoch nur sicher erkennen ob das aktuelle Paket in der erwarteten Reihenfolge relativ zum vorherigen Paket, empfangen wurde. Das ist der Fall, wenn der Wert des aktuellen Pakets mit dem erwarteten Wert übereinstimmt. Der Algorithmus ist in **Listing** 1 dargestellt. Nach dem Ende des Testdurchlaufs stehen in den Variablen *inorder* und $outoforder$ die zwei Messgrößen zur Verfügung.

Listing 1: Sequenznummern Auswertung - Naiv

```
1:  nextSeqNr := 0
2:  outoforder := 0
3:  inorder := 0
4:  procedure PRECESSSEQNR(seqNr)
5:      if seqNr = nextSeqNr then                ▷ Reihenfolge stimmt
6:          inorder++
7:      else                                     ▷ Reihenfolge falsch
8:          outoforder++
9:      end if
10:     nextSeqNr = seqNr + 1
11: end procedure
```

Ist die Sequenznummer größer als erwartet sind folgende Ursachen möglich: Die Pakete der fehlenden Sequenznummern wurden vom DuT verworfen, oder die Reihenfolge der Pakete wurde verändert, das Paket ist zu früh. Falls der aktuelle Wert der Sequenznummer jedoch kleiner als der erwartete Wert ist, kann das zwei Ursachen haben: Das DuT hat die Reihenfolge der Pakete verändert und das aktuelle Pakete ist zu spät oder es wurde ein älteres Paket erneut durch das DuT versendet. Für die erweiterte Auswertung der Sequenznummern werden die folgenden Metriken definiert:

inorder zählt die Pakete, die zusammenhängend und in der richtigen Reihenfolge empfangen wurden.

repeated zählt alle Pakete, die mehrfach aufgetreten sind.

early zählt alle Pakete, die zu früh empfangen wurden.

late zählt diejenigen Pakete, die zu spät empfangen wurden.

maxLateness gibt den größten beobachteten Abstand eines verspäteten Pakets zu dem erwarteten Paket an.

maxEarliness gibt den größten beobachteten Abstand eines verfrühten Pakets zu dem erwarteten Paket an.

paketsLost ist die Anzahl der Sequenznummern, die nicht empfangen wurden.

Listing 2: Sequenznummern Auswertung - Erweitert

```
1:  nextSeqNr := 0
2:  seqNrSeen[ ] = 0                          ▷ Liste gesehener Sequenznummern
3:  repeated := 0
4:  early := 0
5:  late := 0
6:  maxEarliness := 0
7:  maxLateness := 0
8:  procedure processSeqNr(seqNr)
9:      if seqNr == nextSeqNr then                          ▷ Reihenfolge stimmt
10:         seqNrSeen[seqNr] := true
11:         inorder++
12:         nextSeqNr := seqNr + 1
13:     else if seqNr > seqNrNext then          ▷ zu früh oder Paket verloren
14:         maxEarliness := max(maxEarliness, (seqNr − seqNrNext − 1))
15:         seqNrNext := seqNr + 1
16:         early++
17:         seqNrSeen[seqNr] := true
18:     else                                          ▷ zu spät oder wiederholt
19:         if seqNrSeen[seqNr] == true then
20:             repeated++
21:         else
22:             seqNrSeen[seqNr] = true
23:             maxLateness := max(maxLateness, (seqNrNext − seqNr − 1))
24:             late++
25:         end if
26:     end if
27: end procedure
28: procedure getPaketsLost(frameCount)
29:     pktsLost := 0
30:     for i : 0, frameCount do
31:         if seqNrSeen[i] == false then
32:             pktsLost++
33:         end if
34:     end for
35:     return pktsLost
36: end procedure
```

Um zu entscheiden, ob ein Pakete mehrfach aufgetreten ist, ist es nötig zu wissen, welche Sequenznummern schon einmal empfangen wurden. Damit können wieder-

holte Pakete von verspäteten Paketen unterschieden werden. Der Pseudocode für die erweiterte Sequenznummernauswertung ist in **Listing 2** dargestellt. In dem Feld *seqNrSeen* wird für jedes Paket, das zum ersten Mal empfangen wird, ein Bit gesetzt. Die Funktion *processSeqNr* wird für jedes empfangene Paket mit dessen Sequenznummer aufgerufen. Nach dem Ende des Testdurchlaufs kann mit der Funktion *gerPaketsLost* die Anzahl der verlorenen Pakte ermittelt werden.

3.3.4 Konfiguration des Messsystems

Alle Parameter eines Tests sollen in einer klar strukturierten Form gespeichert werden. Neben den Parametern werden auch Daten zum DuT erfasst. Die Konfiguration besteht aus mehreren Abschnitten: Der erste Abschnitt enthält die Informationen über das DuT, die für das Erzeugen des Messprotokolls nötig sind. Im zweiten Abschnitt wird festgelegt, welche Tests durchgeführt werden sollen. Darauf folgt für jeden aktivierten Test ein Abschnitt, der die Parameter des Tests enthält. Der letzte Abschnitt definiert alle Eigenschaften der Testpakete.

3.3.5 Beschreibung der Ablaufsteuerung

Phasen des gesamten Testlaufs

Ein vollständiger Testlauf besteht aus sieben Phasen: In der ersten Phase wird die Testkonfiguration eingelesen und auf logische Fehler überprüft. Falls die Konfiguration konsistent ist, werden in Phase zwei die Streams aus der Konfiguration erzeugt. Damit ist nun bekannt wie viel Queues auf welchen Schnittstellen benötigt werden. In der dritten Phase werden die Schnittstellen initialisiert und die Memory Polls angelegt. Ab diesem Zeitpunkt können Pakete versendet werden. Die vierte Phase ist für die Auflösung der IP Adressen in MAC Adressen verantwortlich. Weil ARP Nachrichten während des gesamten Testlaufs beantwortet werden müssen, wird dafür eine eigener Thread gestartet. Wenn alle IPs aufgelöst werden konnten, wird in der fünften Phase jeder aktivierte Einzeltest durchgeführt. Nach dem Abschluss aller Tests erfolgt in der vorletzten Phase die Nachverarbeitung aller Messergebnisse, anschließend wird mit LaTeX ein Testreport als PDF Datei generiert. In der letzten Phase werden die Schnittstellen wieder freigegeben und das Programm beendet.

Das Suchverfahren

Die Durchsatz und Back-to-Back Tests nach RFC2544 sowie der IMIX Test haben das Ziel, das Maximum der Messgröße x in Abhängigkeit eines oder mehrerer Parameter $\vec{p}_i$ zu ermitteln. Dazu wird der Parameterraum mit einem geeigneten Verfahren durchsucht. Für jeden Suchschritt i im Parameterraum R wird eine Messung M_i durchgeführt, anhand einer Bewertungsfunktion $err(M_i)$ bestimmt das Suchverfahren den nächste Punkt $\vec{p}_i$ im Parameterraum. Unter der Annahme, dass die Bewertungsfunktion monoton steigend für den Parameterraum ist, wird das absolute Maximum gefunden. Für die drei oben genannten Tests wird die Paketverlustrate P_{loss} nach Gleichung 5 (S. 12) als Bewertungsfunktion genutzt. Als Suchverfahren wird für alle drei Tests die binäre Suche eingesetzt, da diese, mit einer Komplexität von $\mathcal{O}(\log n)$, sehr effizient ist. Für den Back-to-Back Test ist die Burstlänge N_{burst} der Parameter, Durchsatz- und IMIX-Test werden über die Ethernet Datenrate D_{eth} gesteuert. Bei bidirektionalen Messungen wird der Parameterraum zweidimensional, für jede Richtung wird parallel und unabhängig nach dem Maximum gesucht. Damit beide Suchen konvergieren ist es erforderlich, dass die Leistungsfähigkeit des DuT pro Richtung unabhängig von der Auslastung der Gegenrichtung ist. Das ist bei komplexen DuT, wie VPN-Gateways, jedoch meist nicht der Fall. Besonders bei schwächeren Systemen können die ein oder zwei verfügbaren CPU Kerne nicht exklusiv zugeordnet werden. Um auch diese DuT vermessen zu können wird ermöglicht, dass für beide Richtungen der gleiche Parameter Wert verwendet wird. Damit ist der Datenstrom symmetrisch. Die gesamte Fehlerfunktion ergibt sich durch verknüpfen der Fehlerfunktionen beider Richtungen. Dieses Vorgehen wird als *bound-Test* bezeichnet, da bei Richtungen aneinander gebunden sind.

Synchronisation

Für jeden Trail werden RX und TX Threads aus dem Hauptprozess gestartet. Um die korrekte Abfolge der Testphasen sicherzustellen, ist eine Synchronisierung der Threads nötig. Dieser Ablauf ist in **Abbildung 10** dargestellt. Der erste Synchronisationspunkt(SyncPt.1) wird nach dem Senden der Warmup-Pakete im TX Thread erreicht, der RX Thread verwirft ab diesem Zeitpunkt für zwei Sekunden alle Pakete und initialisiert anschließend die Auswertung für die nun folgenden Pakete. Der zweite Synchronisierungspunkt (SyncPt.2) ist erreicht, wenn der RX Thread bereit für die Messung ist, ab diesem Zeitpunkt beginnt der TX Thread mit dem Senden. Als Synchronisationsprimitive werden zwei Barrier eingesetzt (br-warmup und br-ready). Diese sind zu Beginn mit der Anzahl der zu synchronisierenden Threads

initialisiert. Der Barrier blockiert in jedem Aufruf der *wait()* Funktion bis alle beteiligten diese Funktion aufgerufen haben. Erst dann kehrt der Aufruf gleichzeitig in allen Threads zurück: Die Ausführung ist synchronisiert. Über die Sendedatenrate $D_{intended}$ und die Rahmenlänge L wird die Anzahl der Pakete $N_{txFrames}$ ermittelt, die nötig sind, um für eine Zeit $T_{trailDuration}$ zu senden (siehe Gleichung (8)). Der TX Thread beendet sich, nach dem alle Pakete gesendet wurden. Für den RX Thread muss entweder $T_{trailDuration} + 2\ sec$ verstrichen sein oder die erwartete Anzahl an Paketen wurde empfangen. Der Paketzähler allein reicht hier nicht, da Pakete verloren gehen können.

$$N_{txFrames} = T_{trailDuration} * \frac{D_{intended}}{L * 8 \frac{Bit}{Byte}} \tag{8}$$

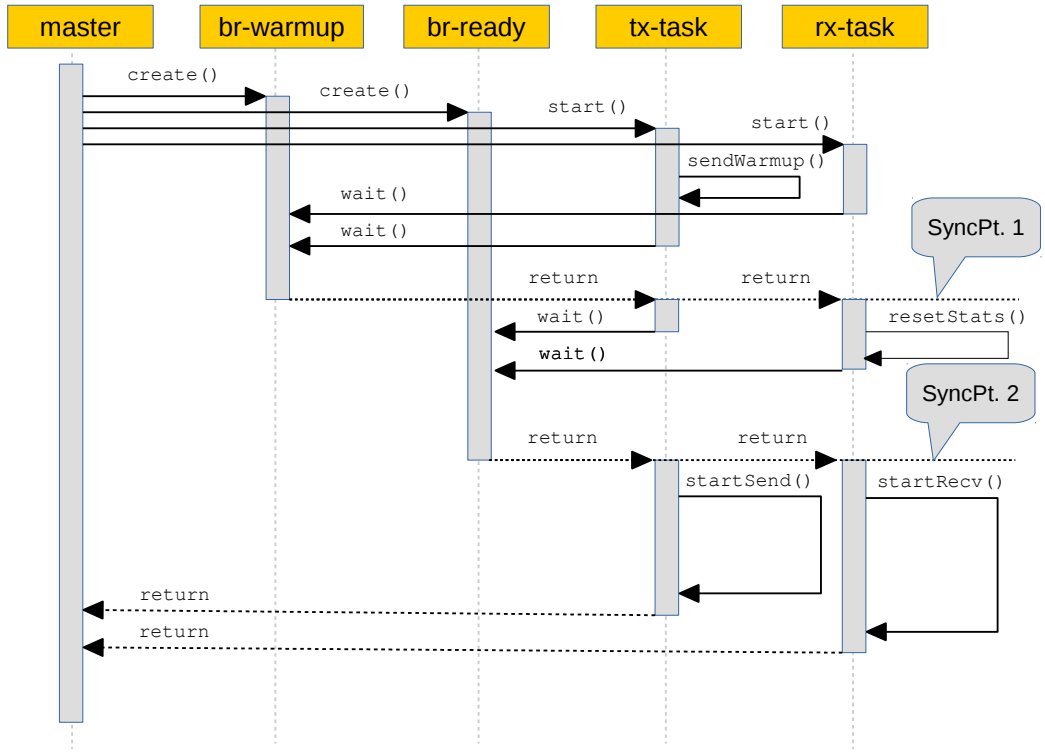


Abbildung 10: Synchronisierung von RX und TX Tasks

4 Umsetzung mit MoonGen

In diesem Kapitel werden alle wichtigen Details der Implementierung des Messsystems erklärt. Der erste Abschnitt befasst sich mit der, aus dem Aufbau des Rechnersystems gegebenen, NUMA Architektur und den Maßnahmen, um diese möglichst effizient zu nutzen.

4.1 Erweiterung der Stream Klasse

Ein Stream Objekt bündelt alle Informationen, die für das Senden und Empfangen eines Datenstroms nötig sind. Die Implementierung erfordert neben den in Abschnitt 3.3.1 aufgeführten Informationen weitere Daten, die für die Ausführung des Messsystems nötig sind. Da ein Stream die Quelle und das Ziel eines Datenstroms darstellt, werden auch die zugeordneten Queues, Mempools sowie CPUs für die TX und RX Seite gespeichert. Weiterhin erhält jeder Stream eine eindeutige Kennnummer, die Stream-ID, welche für die RX Task von Bedeutung ist.

4.2 NUMA Konzept im Datenpfad

Da das Rechnersystem, wie in Abschnitt 3.2 dargestellt, aus zwei CPUs mit jeweils dedizierten Speicher-Subsystemen besteht, handelt es sich um ein NUMA System. Das bedeutet, die Dauer eines Speicherzugriffs hängt von Ziel und Ursprung ab. Daher ist es wichtig, dass ein Thread, dessen Speicher, sowie die Ringpuffer der Netzwerkschnittelle, die der Thread benutzt, auf dem selben NUMA Knoten platziert werden. Das DPDK Framework bietet hierfür vollständige Unterstürzung. MoonGen übergibt jedoch in allen Funktionsaufrufen die Konstante *SOCKET_ID_ANY*. Dadurch wird in der DPDK Funktion *rte_malloc_socket()* immer der aktuelle Sockel des rufenden Threads verwendet.

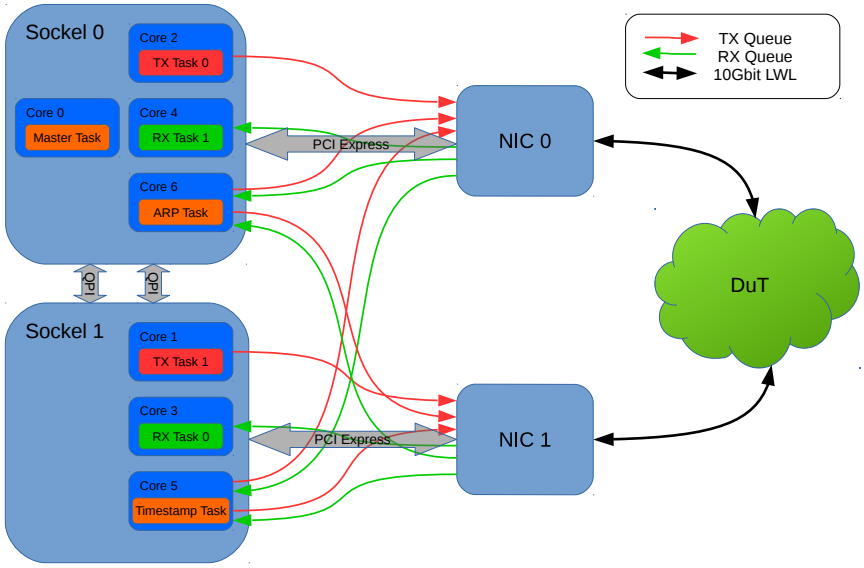


Abbildung 11: NUMA gerechte Task Zuorndnung

Um möglichst jeden Thread und die dazugehörigen Puffer der Netzwerkschnittstelle auf dem lokalen Sockel zu platzieren, wird folgendermaßen vorgegangen: Für jede TX und RX Task wird der Sockel der zugehörigen Schnittstelle bestimmt und bei der Initialisierung der TX und RX-Queues sowie der zugehörigen Memory Pools übergeben. Um den jeweiligen Thread ebenso auf dem passenden Sockel zu platzieren, muss zuerst ermittelt werden, welche CPU-IDs zu welchem Sockel gehören. Dazu wird im MoonGen dpdk Modul [1] eine assoziatives Array angelegt, in dem für jeden Sockel die CPU-IDs gespeichert werden. Für jede RX und TX Task wird aus diesem Array eine CPU des entsprechenden Sockels entnommen. Somit ist sichergestellt, dass CPUs exklusiv verwendet werden. Weil die Tasks für ARP und die Latenzmessung keine hohen Paketraten erreichen müssen, ist es nicht kritisch, wenn diese einem entfernten Sockel zugeordnet werden. Da beide Task mehrere Schnittellen bedienen können, und diese nicht zwingend mit dem gleichen Sockel verbunden sind, ist eine optimale Zuordnung ohnehin nicht immer möglich. Deshalb werden zuerst die CPUs der TX/RX Tasks zugeordnet, erst danach wird aus den restlichen CPUs je eine für ARP und Latenzmessung belegt. In **Abbildung 11** ist die Verteilung der Tasks auf die CPUs für einen Vollduplex-Test dargestellt. Die Nummer der TX- und RX-Task stellt die Zuordnung zu einem Stream dar.

[1] ./lua/include/dpdk.lua

4.3 Der Datenpfad

4.3.1 Funktionsweise der TX Tasks

Für die Tests nach RFC2544 sowie den Durchsatztest mit IMIX Datenstrom wurden drei spezialisierte Senderoutinen entwickelt. Der Ablauf und die Synchronisation mit der RX-Task ist in allen drei Ausprägungen gleich. Es ist jedoch für den IMIX sowie Back-to-Back Test erforderlich, die Lastpakete anders zu generieren, beziehungsweise diese nicht in Blöcken zu versenden. Im folgenden Abschnitt wird die allgemeine TX Task *txLoadTask()* erklärt, darauf folgen die Unterschiede der zwei weiteren Senderoutinen.

Standard TX Task

Die TX Task bekommt unter anderem die Sendedatenrate ($D_{intended}$) und Rahmenanzahl $N_{txFrames}$ sowie ein Stream Objekt beim Start übergeben. Direkt nach dem Start der Task wird die TX Datenrate der Queue gesetzt. Mit den Informationen des zugeordneten Streams werden alle Puffer des Mempools mit Headern der Ethernet, IP und UDP Protokolle gefüllt. Entsprechend der geforderten Paketlänge wird im Anschluss an die Header Informationen der Nutzdatenteil des UDP Pakets geschrieben. Damit sind alle Puffer vorbereitet und müssen nicht vollständig zur Laufzeit des Test geschrieben werden, das ist möglich, da gesendete Puffer wieder im Mempool verfügbar sind und somit wiederverwendet werden.

Das Senden der Pakete erfolgt immer in Blöcken zu N_{busrt} Rahmen, um den Aufwand des Hardwarezugriffs auf mehrere Pakete zu verteilen. Somit ist die Dauer eines Testlaufs, in Paketen gemessen, immer ein vielfaches der Blocklänge. Es wird zunächst ein Block Puffer aus dem Mempool entnommen und für jeden Puffer die Sequenznummer in den ersten acht Byte der UDP Nutzdaten geschrieben. Die Sequenznummer wird für jeden Puffer um eins erhöht und ist 64 Bit lang. Damit ist es möglich für etwa 39 289 Jahre ununterbrochen mit der höchsten Paketrate (14,88 Mpps bei 10 GBit/s) zu senden, ohne das der Zähler überläuft. Nachdem alle Puffer mit einer Sequenznummer versehen sind, wird der Block über die TX Queue des Streams versendet. Dieser Vorgang wird wiederholt, bis $N_{txFrames}$ gesendet wurden. Nach dem die vorgegebene Anzahl an Paketblöcken versendet wurde, wird die TX Task beendet und die gemessene Senderate zurück an den master-Thread gegeben.

BackToBack TX Task

Für den BackToBack Test ist es erforderlich eine beliebige Rahmenanzahl zu senden. Die Standard TX Task arbeitet jedoch blockweise und sendet vor dem Lastdatenstrom WarmUp Pakete, welche für den BackToBack Test nicht erwünscht sind. Es wäre möglich gewesen die Standard TX Task zu erweitern, um jedoch die Komplexität nicht unnötig zu steigern, wurde davon abgesehen. Eine weitere TX Task (*txLoadTaskIndividual(...)*), die die nötige Funktionalität bietet, wurde implementiert.

IMIX TX Task

Der IMIX-Test erfordert, dass die Längen der gesendeten Rahmen dem IMIX Schedule entsprechen. Daher sind die beiden vorherigen TX Task ungeeignet, da immer mit konstanter Rahmenlänge gesendet wird. Eine weiter TX-Task (*txLoadTaskIMIX(...)*) wurde geschrieben. Um jedem Paket, das gesendet wird, eine Position im IMIX Schedule und damit eine Länge zuzuordnen, wird der Index in den Schedule aus der Sequenznummer abgeleitet. Der Index wird durch die Sequenznummer modulo[2] der Schedule Länge bestimmt. Damit ist jeder Sequenznummer eindeutig eine Länge zugeordnet.

TX Raten Kontrolle

Für die Begrenzung der Senderaten kann für jede TX Queue eine individuelle maximale Datenrate gesetzt werden. Eine Kontrolle der erzielten TX Datenraten ist jedoch nötig, da nicht garantiert werden kann, dass die eingestellte Datenrate D_{target} auch erreicht wird. Der Scheduler des Betriebssystems oder zu geringe Leistungsfähigkeit der CPU kann zu Unterbrechungen im Programmfluss oder zu geringer Abarbeitungsgeschwindigkeit führen. Weiterhin ist es durch den Aufbau des Rechnersystems nicht möglich, alle Ressourcen exklusiv zu benutzen: Beispielsweise die Speicherschnittstellen und der PCIe Bus werden von mehreren Tasks des Messsystems und Prozessen des Betriebssystems benutzt. Es ist daher nötig, während des Sendens zu messen, wie hoch die tatsächlich gesendete Datenrate ist. Die Netzwerkschnittstellen bieten hier zwei Möglichkeiten: Es werden auf Port und Queue Ebene Zähler bereitgestellt, die die Anzahl der gesendeten und empfangenen Bytes sowie

[2] Divisionsrest

Rahmen zählen. Für portbasierte Zähler bietet MoonGen schon eine fertige Implementierung, diese wurde um die queuebasierten Zähler erweitert. Die Zählerstände werden jede Sekunde ausgelesen, dadurch wird das Register automatisch auf Null gesetzt. Aus den so erhaltenen Messwerten lässt sich die durchschnittliche Datenrate D_{real} ermitteln. Das exakte Zeitintervall wird durch Auslesen des Time Stamp Counter (TSC) bei jeder Messung ermittelt und ist dadurch sehr präzise (sieht [INT2016] Chapter 17.15). Falls D_{real} außerhalb des Toleranzbereichs $D_{target} \pm D_{txError}$ liegt ist der Trail nicht zwangsweise fehlgeschlagen. Die Entscheidung wird daher in der Ablaufsteuerung des jeweiligen Tests getroffen. Ein weiterer Grund zur Überwachung der Senderate ist ein nicht näher beschriebenes Problem, bei kleinen Paketen und Datenraten größer 58 % Linkrate. In einem Kommentar im MoonGen Code, der später wieder entfernt wurde, weil das Problem nicht nachstellbar war [3], wurde darauf hingewiesen. Hierzu werden im Rahmen der Evaluation des Messsystems genauere Untersuchungen vorgenommen.

4.3.2 Funktionsweise der RX Task

Sequenznummernauswertung

Die Auswertung der Sequenznummern, wie in Abschnitt 3.3.3 vorgestellt, ist die Kernkomponente der RX Task und aufgrund des hohen Rechenaufwands und Speicherbedarfs, besonders zu optimieren. Der Algorithmus beruht darauf, dass bekannt ist, welche Sequenznummern bereits empfangen wurde. Um das zu erreichen, wird ein Bit-Feld mit der Länge der gesendeten Frames herangezogen. Jedes Bit entspricht damit direkt einer Sequenznummer. Die Implementierung wurde aus Optimierungsgründen[4] in C geschrieben, da Lua-JIT keine Unterstützung für Bit-Felder bietet. Ausgehend von einer Traillaufzeit von 600 Sekunden und 14,88 Mpps werden 8,928 Milliarden[5] Sequenznummern gesendet. Das ergibt einen Speicherbedarf von 116 MB [6] für die C Implementierung oder etwa 1173 GB [7] für die Lua Variante. Diese Werte beziehen sich auf einen Datenstrom und sind daher für einen Duplex Test zu verdoppeln. Da Lua-JIT maximal 2 GiB Speicher benutzen kann, ist es somit nicht möglich, die Auswertung in Lua umzusetzen.

[3] https://github.com/emmericp/MoonGen/commit/8d64f03ce596f87fee04a24ddbe9ea8ca8dc799b
[4] Lua Tables benötigen 8 Byte pro Eintrag(64 mal mehr als ein Bit-Feld) und bieten nicht die nötige Performance
[5] 8 928 000 000 Pakete = 600 Sekunden * 14 880 000 pps
[6] 1116 MB = 600 Sekunden * 14,88 Mpps / 8 Bit/Byte
[7] 1173 GB ≈ 600 s * 14,88Mpps * 128 Byte pro Eintrag

Das Bit-Feld wird als zusammenhängender Speicherbereich realisiert, auf den durch die Funktionen *getBit()* und *setBit()* zugegriffen wird. Der Speicher wird dabei als lineares Feld von 64Bit Werten interpretiert. Daher sind die oberen 58 Bit der Sequenznummer der Index in das Feld und die unteren 6 Bit adressieren die einzelnen Bits des indizierten Elements. Der C Code ist in Listing 3 dargestellt.

Listing 3: Sequenznummern Auswertung - Bitfeld Operationen

```
void inline setBit(uint64_t *array, uint64_t bitindex)
{
        array[bitindex>>6] |= (1UL << (bitindex & (0x3f)));
}
uint64_t inline getBit (uint64_t * array, uint64_t bitindex)
{
        return array[bitindex >> 6] & (1UL << (bitindex & (0x3f)));
}
```

Die Sequenznummernauswertung ist als Shared Library realisiert und wird zur Laufzeit in der RX Task über das Lua Fast Function Interface (FFI) API geladen. Es ist erforderlich, dass jede RX Task, beim Aufrufen der Funktionen, eine eindeutige Kennnummer mitliefert um den Sequenznummernkontext zu identifizieren. Das ist nötig, da das gesamte Programm als ein Prozess mit mehreren Threads realisiert ist und daher nur eine Instanz der Bibliothek im Adressraum existiert. Nach dem Start der RX Task wird ein Sequenznummernkontext mit der StreamID als Identifikator und der Anzahl der erwarteten Frames ($N_{txFrames}$) initialisiert. In diesem Schritt wird der nötige Speicher für das Bit-Feld initialisiert und die diversen Zähler dieses Kontextes auf Null gesetzt. Mit der Funktion *processSeqNr(int StreamID,uint64_t seqNr)* wird im Anschluss die Sequenznummer für jedes empfangene Paket durch den in Listing 2 vorgestellten Algorithmus ausgewertet. Nachdem der Trail abgeschlossen ist, wird mit der Funktion *getLost(int StreamID, uint64_t frameCount)* (siehe Listing 4) die Anzahl der Pakete ermittelt, die nicht empfangen wurden.

Für IMIX Datenströme ist es, ohne extra Aufwand während des Empfangs möglich, durch den in Abschnitt 4.3.1 beschriebenen Zusammenhang zwischen Sequenznummer und Paketlänge, verlorene Pakete nach Rahmenlänge zu klassifizieren. Dazu werden im ersten Schritt alle nicht gesetzten Bits (fehlende Sequenznummern) auf ihre Indexposition im Schedule abgebildet und unter dieser aufaddiert (siehe **Listing 5**). Um eine Abbildung dieser Summen auf die ursprünglichen Rahmenlängen zu erreichen, wird der Summen-Vector an die RX Task übergeben. Dort wird, mit

Listing 4: Sequenznummern Auswertung - Paketverlust

```
uint64_t getLost(int StreamID, uint64_t frameCount)
{
        uint64_t temp = 0;
        uint64_t lost = 0 ;
        uint64_t limit = min(rx_ctrs[StreamID].bits,frameCount);

        for(temp= 0;temp<limit;temp++)
        {
                if(!getBit(rx_ctrs[StreamID].bitarray,temp)) lost++;
        }
        return lost;
}
```

Hilfe des Schedules, die Abbildung der Schedule Position auf die Rahmenlängen durchgeführt (siehe **Listing 6**). Als Ergebnis erhält man die Anzahl der fehlenden Pakete, aufgeschlüsselt nach der Rahmenlänge. Da bekannt ist, wie viel Rahmen pro Länge gesendet wurden, lässt sich trivial der relative Paketverlust pro Längenklasse ermitteln.

Listing 5: Sequenznummern Auswertung - Paketverlust IMIX C Teil

```
uint64_t * getHistoFromSchedule(int index, uint64_t schedule_length)
{
        uint64_t temp = 0;
        rx_ctrs[index].histo = malloc(schedule_length * sizeof(uint64_t));
        memset(rx_ctrs[index].histo,0, (schedule_length * sizeof(uint64_t)));
        for (temp= 0;temp<rx_ctrs[index].bits;temp++)
        {
                if(!getBit(rx_ctrs[index].bitarray,temp ))
                        rx_ctrs[index].histo[temp % schedule_length]++;
        }
        return  rx_ctrs[index].histo ;
}
```

Listing 6: Sequenznummern Auswertung - Paketverlust IMIX Lua Teil

```
pktSizeHistogram =  histo:new()
local histo=rx.getHistoFromSchedule(id,imixCfg.parts)
for i=0 ,imixCfg.parts-1 do
        pktSizeHistogram.histo[imixCfg.schedule[i]] = tonumber(histo[i])
end
```

4.4 Latenzmessung

Für den Latenztest nach RFC2544 ist gefordert, dass ein Test 120 Sekunden laufen soll, nach 60 Sekunden muss ein Rahmen markiert werden, um so die Zeitstempel des Sendens und Empfanges ermitteln zu können. Dieser Vorgang soll mindestens 20 mal wiederholt werden um einen Mittelwert zu bilden. Die Datenrate soll dem Ergebnis des Durchsatz-Tests gleichen. Hier weicht die Implementierung von der Vorgabe des RFC2544 ab: Die Testlaufzeit ist konfigurierbar und die Rahmen für die Latenzmessung werden häufiger und während des gesamten Tests gesendet. Es ist kein negativer Effekt auf die Qualität der Messung zu erwarten. Durch die größere Anzahl der Messungen ist das Gegenteil der Fall. Das von MoonGen implementierte Messverfahren sendet Datenpakete, die durch den PTP Header von der Netzwerkschnittstelle erkannt werden. Durchläuft ein solches Paket den TX oder RX Pfad des Chips, wird zum spätest- beziehungsweise frühestmöglichen Zeitpunkt der Wert eines frei laufenden Zählers in je einem Register gespeichert. Da nur ein Register für RX und TX Zeitstempel pro Schnittstelle vorhanden ist, muss auf den Empfang des Pakets gewartet werden, bevor ein weiteres versendet werden kann. Um die Messungen nicht durch den Verlust eines Pakets zu blockieren, wird nach einer Wartezeit von 15 ms[8] vom Verlust des Pakets ausgegangen, es konnte keine Laufzeit ermittelt werden. Somit ist die Paketrate der Latenzmessung direkt von der Laufzeit der Pakete abhängig und wird durch die Wartezeit nach unten begrenzt. Um eine Obergrenze zu gewährleisten, wird die TX Queue per TRS auf 3000 pps begrenzt, das ist möglich, da die Paketgröße konstant ist und somit eine Datenrate ableitbar ist. Eine Begrenzung ist nötig, weil für den Test gefordert ist, dass die Datenrate der des Durchsatz-Tests entspricht. Diese setzt sich für den Latenztest aus der Summe von Lastdatenstrom und dem Datenstrom der Latenzmessung zusammen. Für letzteren ist jedoch nur eine Ober- und Untergrenze bekannt, daher wird der Lastdatenstrom durch ein iteratives Verfahren eingestellt.

[8] 15ms ist der Standardwert den MoonGen nutzt

Die Datenrate des Laststroms D_{load} wird initial nach nach **Gleichung** (9) berechnet. In jedem Iterationsschritt wird die erzielte Gesamtdatenrate D_{real} mit der Zieldatenrate D_{target} verglichen, wenn D_{real} im Intervall $[D_{traget} - errLimit, D_{target}]$ liegt, wird überprüft, ob die eingestellte Paketverlustrate nicht überschritten wurde und die Iteration beendet. Durch den Schwellwert *errLimit* wird ein Unterschreiten der Datenrate gestattet, ein Überschreiten muss jedoch vermieden werden, um die Paketverluste über dem Limit auszuschließen. Der Schwellwert sollte der eingestellten Auflösung des Durchsatz Test entsprechen um die Abweichung der Datenraten möglichst gering zu halten. Falls D_{real} nicht im Intervall liegt, wird D_{load} nach **Gleichung** (10) angepasst und eine neuer Durchgang gestartet. Der Adaptionsschritt entspricht einem Proportional Regler[9] und kann daher nie den eingestellten Zielwert exakt erreichen. Die Auflösung des TRS und das Regelverhalten erzwingen die Nutzung eines Akzeptanzintervalls.

$$D_{load} = D_{target} - frameSize * 8 * 2000pps \tag{9}$$

$$D_{Load} = D_{load} + \frac{D_{target} - D_{real}}{2} \tag{10}$$

4.5 Suchverfahren

Für den Durchsatz, Back To Back und IMIX Test muss ein Suchverfahren implementiert werden, das robust, zuverlässig und möglichst schnell arbeitet. Das Verfahren besteht aus zwei Schritten: im ersten Schritt wird mittels binärer Suche der Parameterraum in Abhängigkeit der Fehlerfunktion durchsucht. Der Zweite Schritt dient der Validierung des gefundenen Ergebnisses. Durch diese Aufteilung ist er möglich, die Traildauer für den ersten Schritt kleiner zu wählen um die Suche zu beschleunigen. Im Validierungschritt wird mit der vollen Traildauer sichergestellt, dass der Wert der Fehlerfunktion kleiner gleich dem vorgegebenen Schwellwert ist. Liegt der Wert über dem Grenzwert, so wird der Parameter um die minimale Schrittweite der binären Suche iterativ verringert, bis der Grenzwert erreicht wird. Das ist einerseits nötig da sich das DuT, bei zu geringer Traildauer, im Suchschritt anders verhalten kann als bei voller Traildauer. Andererseits ist bei den getesteten DuT zu beobachten gewesen, dass die Wiederholung eines Trail, trotz gleichen Parameter, zu unterschiedlichen Werten der Fehlerfunktion führen kann. Durch das zweistufige

[9] P-Regler haben eine bleibende Regelabweichung

Suchverfahren kann somit ein schneller, robuster und zuverlässiger Test gewährleistet werden.

Listing 7: Suchalgorithmus: binäre Suche mit Grenzwert

```
local binarySearch = {}
binarySearch.__index = binarySearch
function binarySearch:create(lower,upper)
        local self = setmetatable({},binarySearch)
        self.lowerLimit = lower
        self.upperLimit = upper
        return self
end
function binarySearch:next(curr,valid,threshold)
        if valid then
                if curr >= self.upperLimit then
                        return curr,true
                else
                        self.lowerLimit = curr
                end
        else
                if curr <= self.lowerLimit then
                        return curr,true
                else
                        self.upperLimit = curr
                end
        end
        local nextVal = (self.lowerLimit + self.upperLimit) / 2
        if math.abs(nextVal - curr) < threshold then
                return curr,true
        end
        return nextVal, false
end
```

Die Implementierung der binären Suche wird ist in **Listing 7** dargestellt. Eine Instanz muss mit der *:create()* Funktion initialisiert werden, die Argumente *upper* und *lower* geben die obere und untere Grenze an. Im Anschluss kann durch den Aufruf der *:next(curr,valid,threshold)* Funktion der nächste Suchschritt bestimmt werden, dabei gibt das Argument *curr* den aktuellen Wert, beziehungsweise im ersten Schritt

den Startwert, des Parameters an. Das Argument *valid* gibt an, ob der Trail für *curr* erfolgreich war, *threshold* ist Untergrenze für die Schrittweite, ab der die Suche als abgeschlossen gilt. Jeder Aufruf liefert anhand der Argumente und des inneren Status der Suche zwei Rückgabewerte, den neuen Parameter sowie einen boolschen Wert, der signalisiert, dass die nächste Schrittweite kleiner gleich *threshold* ist und damit die Suche beendet ist.

4.6 Fehler der verwendeten Softwarekomponenten und deren Lösung

Während der Implementierung sind einige Fehler in MoonGen und DPDK aufgetreten. Im folgenden werden drei Probleme vorgestellt, deren Untersuchung und Lösung größeren Aufwand erforderte. Es ist angestrebt, die Lösungen der gefundenen Fehler in die jeweiligen Softwareprojekte zurückfließen zu lassen.

Einfrieren der ARP Task

Die von MoonGen bereitgestellte ARP Task hat regelmäßig ein fehlschlagen der Messungen verursacht, weil nach einiger Zeit ($\geq$1 h) keine ARP Request des DuT mehr beantwortet wurden. Ursächlich für das Einfrieren war, dass der TX Speicher Pool korrupt war und so keine Pakete mehr versendet werden konnten. Das zugrundeliegende Problem wurde durch die falsche Benutzung der DPDK API durch MoonGen hervorgerufen. Wenn die ARP Task eine IP Adresse auflösen soll, muss ein ARP Request gesendet werden. Dazu wird ein Paket Puffer aus dem TX Pool entnommen und das Paket vorbereitet, im Anschluss versendet die ARP Task den Paket Puffer auf jeder konfigurierten Schnittstelle. Nach dem das Paket versendet ist wird der Puffer automatisch wieder in den Pool gegeben, aus dem er stammt. Das geschieht hier jedoch mit dem selben Puffer mehrmals und verursacht so, dass die Verwaltungsstrukturen des Speicherpools nicht mehr konsistent sind. Ein weiterer Effekt ist, dass durch das Senden auf allen Schnittstellen, auch bei vertauschten Kabeln, ein Auflösen der IP Adressen möglich ist.

Um das Problem zu lösen, muss die ARP Task entscheiden können, auf welcher Schnittstelle ein Request versendete werden soll. Der Code wurde um eine Liste Schnittstellen erweitert und speichert jetzt auch die konfigurierten IP Adressen der jeweiligen Schnittstelle. So kann entschieden werden, ob die aufzulösende IP Adresse

zum Subnetz der jeweiligen Schnittstelle gehört. Dadurch wird der ARP Request nur auf der passenden Schnittstelle gesendet. Durch diese Anpassung ist es nun möglich, länger Testläufe durchzuführen.

Speicher Pool Verwaltung

Weil die selben Speicherpools für den gesamten Testlauf wiederverwendet werden, müssen alle Paket-Puffer für jeden Trail neu geschrieben werden, um die Rahmenlänge, einige Header Informationen sowie den Payload Teil der UDP Pakete anzupassen. Durch die Funktionsweise des Caches der Speicherpools ist es jedoch nur das erste mal möglich, alle Pakete mit einem mal zu reservieren. Der Aufruf der DPDK Funktion `rte_mempool_get_bulk()` schlägt das zweite mal fehl, da sich einige Puffer im Cache befinden und die Funktion die Anfrage entweder aus dem Cache oder Ring des Speicherpools bedient. Der Cache hat jedoch immer weniger Elemente als der gesamte Pool groß ist, dadurch kann eine wiederholte Anfrage aller Puffer nicht funktionieren sobald mindestens ein Element im Cache ist. Das ist jedoch unweigerlich der Fall. Weil MoonGen den Rückgabewert der Funktion stillschweigend ignoriert, äußert sich das Problem erst durch Speicherzugriffsfehler bei der Initialisierung der Puffer. Zur Lösung des Problems wurden zwei Ansätze untersucht: Die erste Option ist, den Cache zu deaktivieren und damit das Problem zu umgehen. Das birgt jedoch die Gefahr, die Leistungsfähigkeit des TX Pfads zu beeinträchtigen. Die zweite Möglichkeit besteht in der Vergrößerung des Caches, damit der gesamte Pool hinein passt und somit die Anfrage nach allen Elementen bedient werden kann. Dazu sind einige Änderungen in der DPDK Bibliothek, deren Konfiguration sowie im MoonGen Code nötig. Trotzt des höheren Aufwands wurde dieser Ansatz gewählt, weil die Vorteile des Speicher Pool Caches weiter genutzt werden können.

RX Filter

Um die RX Filter der Netzwerkkarte einzurichten, bietet MoonGen die Funktion `addHW5tupleFilter()`, welche das DPDK API benutzt, um die Filter zu setzen. Die Filter werden zwingend benötigt, um bestimmte Datenströme auf eine RX Queue zu leiten. Durch eine falsche Reihenfolge der Bytes, mit der die IP und Port Filter Register programmiert werden, funktionierten diese jedoch nicht wie erwartet. Durch verwenden der korrekten Reihenfolge (Big- statt LittleEndian) funktionierten die Filter wie erwartet.

4.7 Dokumentation der Messungen

Nach dem erfolgreichen Abschluss der Messungen werden die Ergebnisse aller Durchläufe jedes Tests zu einem Mittelwert zusammengefasst. Für den IMIX Test wird außerdem die Paketverlustrate nach Rahmenlängen aufgeschlüsselt. Die Dokumentation des gesamtem Messdurchgangs erfolgt auf drei Arten: Für jeden Einzeltest werden die Ergebnisse aller erfolgreichen Trails in je einer Datei gespeichert, um später eine maschinenlesbare Weiterverarbeitung zu ermöglichen. Während der gesamten Laufzeit werden wichtige Informationen in eine Log Datei geschrieben, somit der Testablauf auch später nachvollziehbar. Nach dem Ende aller Tests und dem aufbereiten der Messdaten wird ein Messprotokoll in Form einer Latex Datei generiert. Das Protokoll enthält die Metadaten aus der Testkonfiguration, sowie alle relevanten Parameter der durchgeführten Test. Darauf folgen die gemessenen Werte, sowie Diagramme zur Veranschaulichung der Ergebnisse.

5 Messungen und Messaufbauten

In diesem Kapitel werden die durchgeführten Messungen und die verwendeten Geräte und Aufbauten beschrieben. Das Ziel der Messungen ist es, eine Datenbasis für die Evaluierung des im Rahmen dieser Arbeit entwickelten Messsystems zu erstellen. Dazu sollen für die folgenden Aspekte Daten erhoben werden:

1. Leistungsfähigkeit des Sende-Datenpfads
2. Leistungsfähigkeit des Empfangs-Datenpfads
3. Genauigkeit der nettest Messung gegenüber etablierten Messsystemen

5.1 Beschreibung der Referenzmesssysteme

Es stehen für die Messungen zwei etablierte Messgeräte zur Verfügung. Beide Geräte nutzen für die Paketverarbeitung spezielle FPGA Einschubkarten und sind dadurch in der Lage, auch bei kleinsten Rahmenlängen maximale Datenraten zu erzeugen und die empfangenen Rahmen auszuwerten. Die Messsysteme gelten als sehr genau und zuverlässig, daher sind sie für vergleichende Messungen geeignet.

5.1.1 Spirent Testcenter N4U

Das Spirent Testcenter N4U ist ein modulares Messsystem das für den Einbau in 19 Zoll Netzwerkschränken vorgesehen ist. Mit seinen 4 Höheneinheiten bietet es Platz für 2 Moduleinschübe. Über die Module kann das Gerät mit diversen Schnittstellen und Funktionen ausgerüstet werden. Das zur Verfügung stehende Gerät ist mit einem fX2-40G-Q5 Module bestückt und bietet fünf Quad Small Form-factor Pluggable (QSFP+) Steckplätze. Jeder der 5 Ports kann mit 40 GBit/s betrieben werden. Durch die Verwendung eines Adapterkabels lässt sich ein QSFP+ Steckplatz in vier

SFP+ mit je 10 GBit/s aufteilen. Das fX2-Modul ist für Messungen und Tests auf den Protokollebenen eins bis drei des OSI-Models ausgelegt und erreicht für Rahmenlängen von 58 Byte bis 16383 Byte die maximal möglichen Paketraten. Für Zeitmessungen wird eine Auflösung von 2.5 ns sowie eine Genauigkeit von 10 ns angegeben [SPI]. Neben den Standard-Protokollen Ethernet, IP und UDP werden diverse andere Technologien und Protokolle unterstützt. Die Testcenter Plattform von Spirent wird über eine dedizierte Ethernet Verbindung von einem Arbeitsplatz PC gesteuert und ermöglicht das parallele arbeiten mehrere Benutzer. Da das verbaute Schnittstellenmodul nur 40 und 10 GBit/s Schnittstellen bietet, können nicht alle *Device Under Test* (DuT) angeschlossen werden.

5.1.2 EXFO FTB-400

Der FTB-400 der Firma EXFO ist ein tragbares Netzwerk- und Verkablungsmessgerät. Durch die Integrierte Benutzerschnittelle kann das Gerät auch im Feld eingesetzt werden. Das Messgerät bietet Platz für 2 Schnittellenmodule, wodurch eine flexible Anpassung an die jeweiligen Umgebungen möglich ist. Der verfügbare FTB-400 hat das Module FTB-8510 verbaut, welches für Ethernet Tests ausgelegt ist. Es bietet zwei Schnittstellen die bis 1 GBit/s betrieben werden können. Jede Schnittstelle bietet einen RJ45 Kupfer Port, der Ethernet in 10,100 und 1000 MBit/s unterstützt. Außerdem kann alternativ ein Small Form-factor Pluggable (SFP) Modul gesteckt werden um LWL Konnektivität zu ermöglichen. Das installierte Softwarepaket bietet die Möglichkeit, alle vier Tests des RFC2544 Standards durchzuführen, IMIX Tests werden nicht unterstützt.

5.2 Messaufbauten

5.2.1 VPN

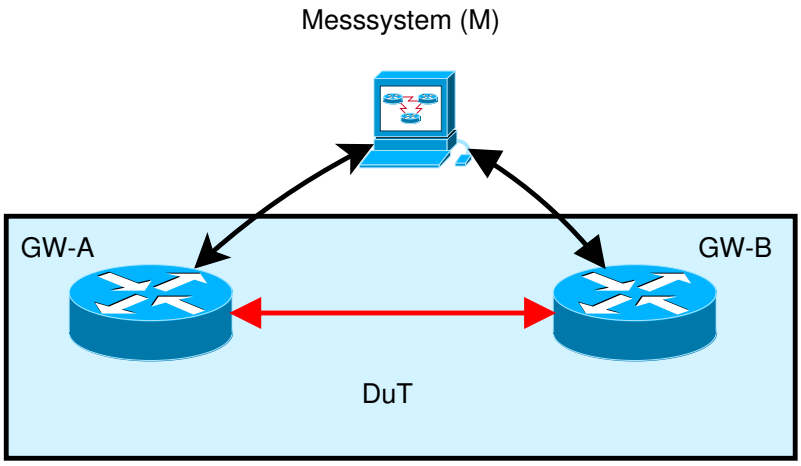


Abbildung 12: Messaufbau VPN

Der Messaufbau VPN ist ein typischer Aufbau um VPN-Gateways und Router zu vermessen. Es werden zwei gleiche Systeme (GW-A, GW-B) direkt verbunden, an je einer weiteren Schnittstelle wird das Messsystem angeschlossen. Es werden zwei Softwarevarianten getestet, *Linux* bezeichnet ein GNU/Linux System, das die Messdaten verschlüsselt durch einen IPsec Tunnel leitet. Mit *pfSense* wird ein FreeBSD System bezeichnet, das als Router konfiguriert ist. Der IPsec Tunnel wird im Tunnelmode mit AES192-CBC/SHA256 betrieben und hat eine Schlüssellebenszeit von 10 Tagen gesetzt. Die Zeit sollte länger als ein Testdurchlauf sein, da eine Neuaushandlung des Tunnels während der Messung zu Paketverlusten führen würde und dadurch die Ergebnisse verfälschen würden. Der schematische Aufbau ist in **Abbildung 12** dargestellt. Für die Messungen stehen 2 Paar unterschiedlich leistungsstarke Rechnersysteme zur Verfügung (siehe **Anhang B Tabelle 19**). Das Ziel dieses Aufbaus ist es, die selben DuT mit verschieden Messsystemen zu vermessen und dadurch einen Vergleich der Ergebnisse zu ermöglichen. Als Messverfahren wird eine vollständige RFC2544 Messung und, soweit anwendbar, eine IMIX-Messung durchgeführt. In **Tabelle 6** sind alle durchgeführten Messungen aufgeschlüsselt. Eine Test mit pfSense war auf DuT 1 nicht möglich, da die Software aufgrund von Hardwareinkompatibilität nicht lauffähig war.

Die Testparameter wurden, soweit möglich, gleich gewählt, um die Vergleichbarkeit der Ergebnisse zu gewährleisten. Dabei ist zu beachten, dass die Traildauer für das

Spirent Testcenter variabel ist, da der Trail abgebrochen wird sobald die festgelegte Paketverlustrate überschritten ist. Damit wird die Gesamtlaufzeit verkürzt, es ist jedoch kein Einfluss auf die Messwerte zu erwarten, weil ein erfolgreicher Trail mit der gesamten Traildauer von 60s durchgeführt wird. Alle Parameter sind in **Tabelle 7** und **Anhang C - Tabelle 20** dokumentiert.

Tabelle 6: Durchgeführte Messungen für Aufbau VPN

Messsystem / **DuT - Software**	**nettest**	**Spirent**	**EXFO**
DuT 1 - Linux	RFC, IMIX	RFC,IMIX	
DuT 2 - Linux	RFC, IMIX		RFC
DuT 2 - pfSense	RFC, IMIX		RFC

Tabelle 7: Testparameter: RFC2544 und IMIX

Messsystem / **Parameter**	**nettest**	**Spirent**	**EXFO**
Traildauer	10 s	dynamisch [1]	10 s
Validierungsdauer	60 s	60 s	60 s
Auflösung	0,1 MBit/s	0,001 % Linerate	0,125 MBit/s
Rahmenlängen	RFC + 1420	RFC + 1420	RFC
Paketverlust Limit	0 %	0 %	0 %
Wiederholungen zur Mittelwertbildung	3	3	3
Frameloss Step	10 %	10 %	10 %
Back to Back Obergrenze	100000	100000	5 s
Back to Back Auflösung	16	16	16
IMIX Profil	Spirent IPSEC	Spirent IPSEC	

[1] Der Trail wird als Fehlerhaft abgebrochen, sobald die Fehlerschwelle überschritten ist. Dadurch kann das Suchverfahren beschleunigt werden.

5.2.2 Direktverbindung

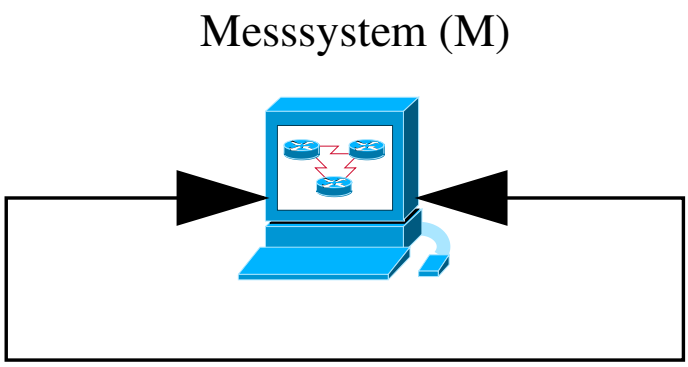


Abbildung 13: Messaufbau Direktverbindung

Für den Aufbau ”Direktverbindung”werden zwei Schnittellen des Messgeräts direkt miteinander verbunden. Der Aufbau ist in **Abbildung 13** schematisch dargestellt. Durch das passive Kabel wird, bis auf sehr geringe Signallaufzeiten, kein Einfluss auf die übertragenen Pakete genommen. Damit können alle Paketverluste, die nicht durch das Messsystem selbst verursacht werden, ausgeschlossen werden. Somit ist es möglich, die paketverarbeitenden Komponenten des Messsystems selbst zu testen. Um die Leistungsfähigkeit der TX und RX Task zu ermitteln, wird ein Durchsatz Test nach RFC2544 sowie eine IMIX Durchsatzmessung durchgeführt. Die Messungen werden für die Kupfer- sowie LWL-Schnittstellen und die in **Tabelle 8** aufgelisteten Sockel/NIC Kombinationen durchgeführt. Die Spalte CPU Sockel gibt dabei an, mit welcher CPU die Schnittstelle verbunden ist.

Tabelle 8: Schnittstellen und Sockel Kombinationen für Aufbau ”Direktverbindung”

Messung	NIC 0	NIC 1	CPU Sockel NIC0	CPU Sockel NIC1
1	X520-T2 Port0	X520-T2 Port1	0	0
2	X520-T2 Port0	X520-T2 Port1	1	1
3	X520-SR	X520-SR	0	1
4	X520-SR	X520-SR	1	1

Weiterhin wird die Genauigkeit der erzeugten Datenraten durch eine Untersuchung des TRS bestimmt. Dazu wird die eingestellte Datenrate D_{target} mit der gemessenen Senderate D_{real} verglichen und die Abweichung $D_{error} = D_{real} - D_{target}$ bestimmt. Der Parameterraum setzt sich aus der Rahmenlänge sowie der Datenrate zusammen. Für die Messungen wurden zwei Ports des Messsystems direkt verbunden, ein

MoonGen Skript durchläuft den Parameterraum und führt die Messungen durch. Als Schrittweite für D_{traget} wurden 25 MBit/s gewählt, das führt zu 400 Messpunkten pro Rahmenlänge. Die Schrittweiten der Rahmenlänge wurden in drei Bereiche gestaffelt, sehr klein, klein und groß (siehe **Tabelle 9**). Aufgrund der in Abschnitt 4.3.1 erwähnten Probleme sind besonders die sehr kleinen Längen von Interesse.

Tabelle 9: Rahmenlängen für TRS Messung

Bereich	**Start**	**Ende**	**Schrittweite**
sehr klein	64	85	1
klein	88	138	2
groß	256	1518	64

6 Auswertung der Messergebnisse

In den folgenden Abschnitten werden die wichtigsten Messergebnisse vorgestellt und deren Plausibilität diskutiert. Anschließend erfolgt die Auswertung der Messwerte bezüglich der Vergleichsmessungen und der erwarteten Leistungsfähigkeit. Die vollständigen Messwerte sind in **Anhang D**, nach DuT gruppiert, aufgeführt. Für alle mit *nM* gekennzeichneten Zellen sind keine Messwerte verfügbar. Insbesondere die Frameloss Messungen mit dem EXFO Messsystem waren nicht verwertbar, da kein sinnvoller Export der Messergebnisse möglich ist.

6.1 Leistungsfähigkeit der Paketverarbeitung

Für die Durchsatz Messung wird bei direkt verbundenen Netzwerkschnittstellen kein Paketverlust sowie volle Auslastung der Schnittstelle erwartet. Die Messwerte in **Tabelle 10** zeigen jedoch für 64 Byte Rahmen (RFC2544 Durchsatztest) in Messung 1 und 2 einen geringeren Wert. Das bedeutet, dass in diesen Fällen die verfügbare Leitungskapazität nicht ausgelastet werden konnte.

Das Messsystem erreicht also nicht für alle Kombinationen aus Schnittstellen und Rahmenlängen die volle Datenrate. Es ist keine Abhängigkeit zur Anbindung an eine bestimmten Sockel zu erkennen. Für Messung 1 und 2 geht aus dem Messprotokoll hervor, dass die TX Task mit voller Datenrate gesendet hat, aber auf der Empfangsseite Pakete verloren werden. Wird die TX Datenrate gesenkt, so steigt

Tabelle 10: Direktverbindung - maximale Datenrate / % Linerate

Messung	**Rahmenlänge / Byte**								
	64	128	256	512	1024	1280	1420	1518	IMIX
1	74,33	100	100	100	100	100	100	100	100
2	73,94	100	100	100	100	100	100	100	100
3	100	100	100	100	100	100	100	100	100
4	100	100	100	100	100	100	100	100	100

die RX Datenrate um etwa den selben Betrag an. Das lässt auf eine hardwarebedingte Übertragungsraten-Grenze schließen. Da die Threads für TX und RX auf dedizierten CPU Kernen laufen, ist hier kein gegenseitiger Einfluss zu erwarten. Durch Messung 3 und 4 konnte nachgewiesen werden, dass die Software zwei Netzwerkschnittstellen unter maximaler Last bedienen kann, was die Vermutung einer hardwarebedingten Ursache unterstützt. Der Unterschied zwischen den Messungen 1 und 2 sowie 3 und 4 besteht in den verwendeten Netzwerkkarten. Beide Karten werden per PCI-Express v2 x8 angebunden, es stehen somit 4 GB/s pro Richtung zur Verfügung. Für die Übertragung der Pakete zweier 10 GBit/s Streams sind pro Richtung 1904[1] MB/s nötig, hinzu kommt ein von der Paketrate abhängiger Anteil[2], der für 64 Byte Rahmen mindesten 476[3] MB/s beträgt. Damit ist nur noch etwa ein viertel der theoretischen Übertragungskapazität verfügbar. Die reale Auslastung der PCI-Express Bus ist auf dem verwendeten System nicht messbar, es existieren zwar Performance-Zähler[4] im PCI-Komplex der CPU, deren Bedeutung ist jedoch nur unzureichend dokumentiert (siehe [INT2015] ab S.197). Somit war eine weiter Analyse nicht möglich.

Aufgrund der Einschränkung der Datenrate ist es zu empfehlen, dass die Intel X520-SR1 Schnittstellen gegenüber den X520-T2 vorzuziehen sind, insbesondere für Duplex Messungen, falls zu erwarten ist, dass das DuT mehr als 73 % Linerate bei 64 Byte Rahmenlänge verarbeiten kann. Falls zwingend 10GBASE-T verwendet werden muss, sollten zwei X520-T2 verbaut und nur einer der Ports genutzt werden.

6.2 Genauigkeit der TX Raten Begrenzung

Für die Funktion des Messsystems ist die Senderatenbegrenzung eine zentrale Komponente. In **Abbildung 14** ist der Datenratenfehler $D_{error} = D_{set} - D_{real}$ auf der Z-Achse über der Rahmengröße (X-Achse) und der Eingestellten Datenrate (D_{set}) aufgetragen. Die Datenraten sind auf OSI-Schicht 1 bezogen. Es ist für sehr kleine Rahmenlängen und hohe Datenraten ein erheblicher Fehler von bis zu 952 MBit/s zu beobachten. Zur Untersuchung des Verhaltens sind in **Abbildung 15** nur Messwerte für 64 bis 88 Byte Rahmen eingezeichnet. Es ist eine Grenze erkennbar, ab der der Fehler erheblich ansteigt. Diese Grenze verläuft von 8500 MBit/s für 64 Byte bis 10000MBit/S für 79 Byte. Weiterhin ist in den Fehlerwerten ab etwa 4000

[1] $1904MB \approx 10000MBit/s * \frac{64Byte}{64Byte+20Byte(IFG+Preambel)} * 2Streams$

[2] je Rahmen muss ein Paket Descriptor von 16 Byte übertragen werden

[3] $476MB/s \approx 14.88Mpps * 16Byte * 2Streams$

[4] ermöglichen dass Zählen diverser Ereignisse z.B. Cache Miss/Hit

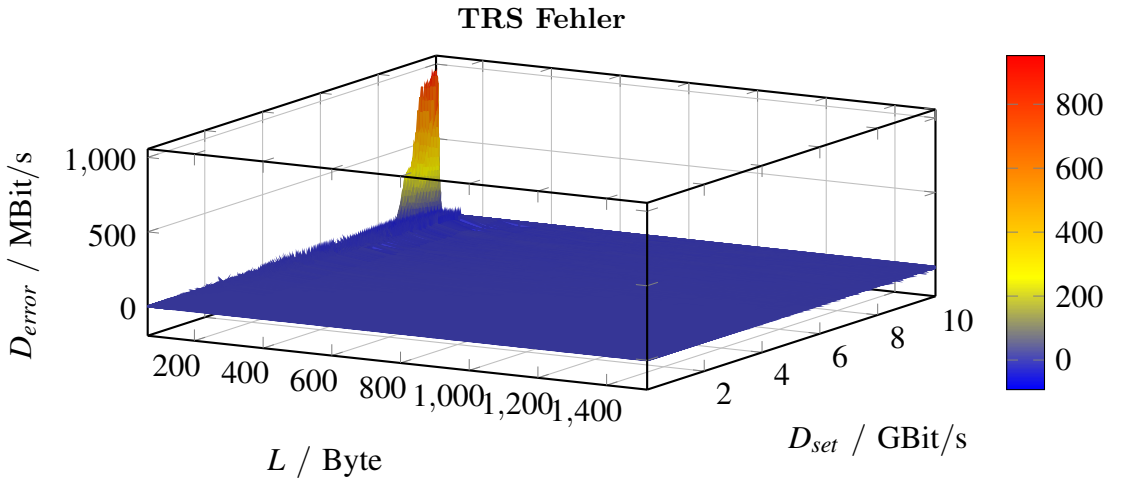


Abbildung 14: TRS Fehler in Abhängigkeit der Layer 1 Datenrate und Rahmenlänge

MBit/s ein wiederkehrendes Muster erkennbar. In **Diagramm 16** sind alle Messungen für Rahmen ab 88 Byte Länge dargestellt. Da alle D_{error} Werte negativ sind ist die Z Achse hier invertiert, um eine besser Sichtbarkeit zu gewährleisten. Allgemein lässt sich erkennen, dass der Fehler mit steigender Datenrate und fallender Rahmenlänge zunimmt. Der Vorzeichenwechsel der D_{error} Werte, ab der zuvor beschrieben Grenze, ist problematisch, da der Suchalgorithmus darauf angewiesen ist, dass ein monoton steigender Zusammenhang zwischen der eingestellten und realen Datenrate existiert. Daher ist für Datenraten, die über der durch $D_{max} = 100 * L + 2100$ festgelegten Grenze liegen, keine Messung möglich.

Die Abweichung der eingestellten zur realen Datenrate ist für die Genauigkeit der Messung im größten Teil des Parameterraums nicht relevant, da die Führungsgröße des Suchalgorithmus die Paketverlustrate ist. Die erreichbare Genauigkeit ist deshalb von der in Abschnitt 2.6.1 erklärten Granularität des TRS abhängig. Damit hängt die Genauigkeit der Durchsatz-, IMIX-, Frameloss-Tests von der Datenrate ab, da die Granularität mit steigenden Datenrate größer wird. Die Obergrenze beträgt somit nach Diagramm 5 0,85 % der maximalen Ethernet Datenrate. Als untere Grenze gibt Intel in der Dokumentation einen *ratefactor* von maximal 1 000 an, was 10[5] MBit/s bei einem 10 GBit/s Ethernet Link entspricht.

[5] $D_{trs} = \frac{D_{Link}}{ratefactor}$

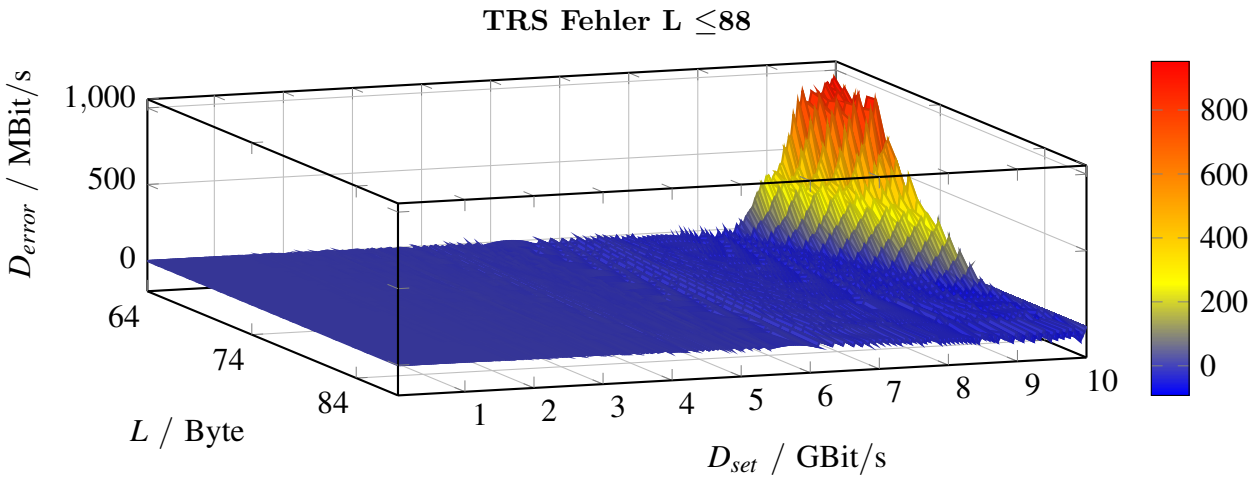


Abbildung 15: TRS Fehler in Abhängigkeit der Layer 1 Datenrate und Rahmenlänge (≤88 Byte)

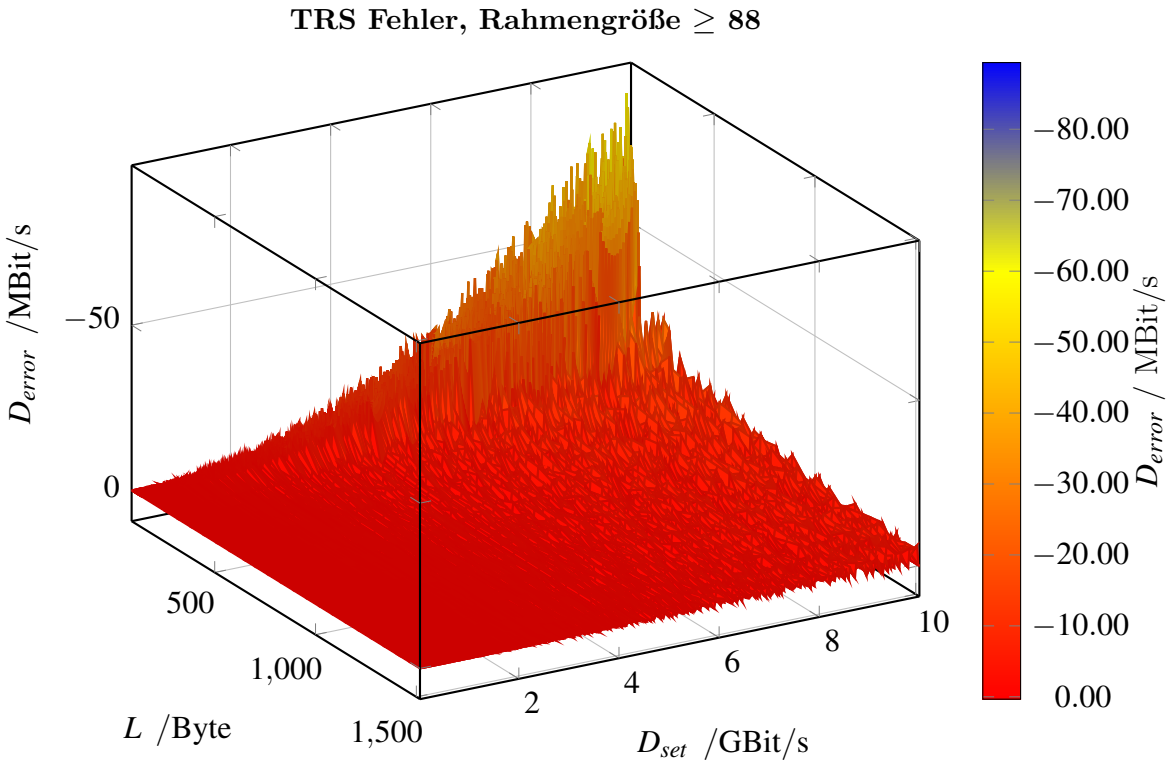


Abbildung 16: TRS Fehler in Abhängigkeit der Layer 1 Datenrate und Rahmenlänge (≥88 Byte)

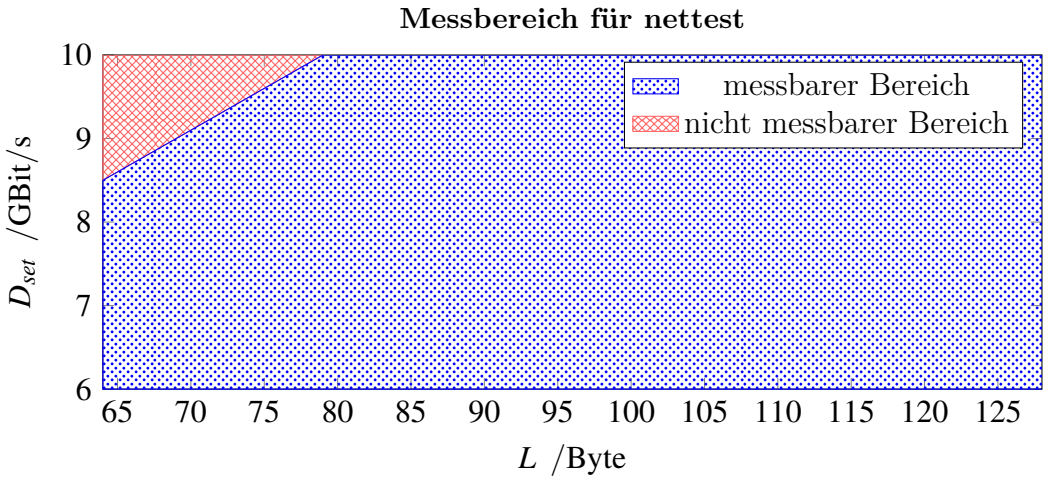


Abbildung 17: Messbereich Datenrate für nettest

6.3 Auswertung der Vergleichsmessungen

6.3.1 Auswertung der Durchsatz Tests

In **Abbildung 18** sind exemplarisch die Messwerte des Durchsatz-Tests von DuT 1 für die Messgeräte nettest ($D_{nettest}$) und Spirent ($D_{spirent}$) dargestellt. Das erwartete Ergebnis sollten für die Rahmenlängen bis einschließlich 1420 Byte eine steigende Datenrate ergeben, der Messwert für 1518 Byte sollte unter dem Maximum liegen. Dieses Verhalten wird durch die Fragmentierung der IP Pakte verursacht. Diese ist nötig, da für den IPSec Tunnel ein IP und ESP Header vor jedes Paket gesetzt wird und damit die maximale Rahmenlänge von 1518 Byte überschritten wird. Da das verschlüsselte IP Paket auf zwei Rahmen aufgeteilt werden muss, entsteht beim Sender und Empfänger einen Mehraufwand, welcher sich negativ auf die Datenrate auswirkt. Die gemessenen Werte beider Messsysteme entsprechen diesem Verlauf und sind somit plausibel.

In **Tabelle 11 S. 64** sind die Ergebnisse des Durchsatz-Tests sowie die absolute (ΔD) und relative (δD) Differenz der Messungen von DuT 1 beider Messsystem aufgelistet. Die relative Differenz bezieht sich auf die maximale Ethernet-Datenrate (D_{maxL2}). Die relativen Abweichungen entsprechen den Erwartungen (TRS Schrittweiten) mit Ausnahme der Rahmenlängen 1024 und 1518 Byte. Eine akribische Überprüfung der Messwerterfassung in nettest ergab kein Fehlverhalten der Software, die Paket- und Bytezähler stimmen exakt mit den erwartete Werten überein.

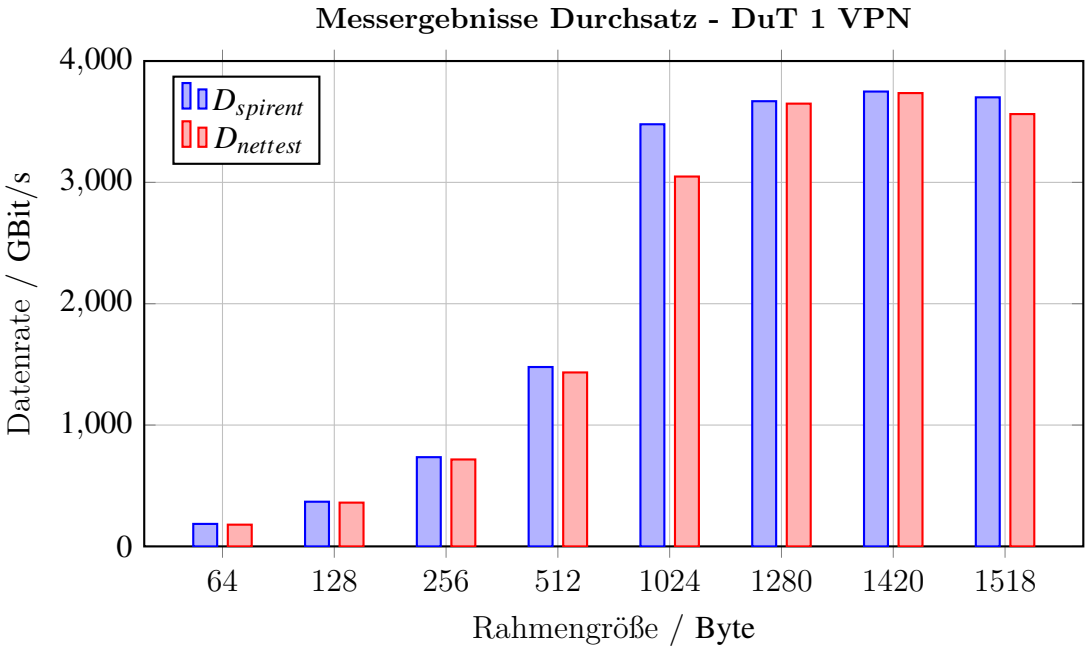


Abbildung 18: Messergebnisse: Datenraten - DuT 1, Linux

Aus dem nettest-Messprotokoll geht hervor, dass bei der, durch den Spirent Tester ermittelten Datenrate, definitiv eine Paketverlustrate ≥ 0 mit nettest gemessen wurde. Da Spirent marktführend im Bereich Netzwerkmesstechnik ist, erscheint auch hier ein Messfehler als sehr unwahrscheinlich. Weil auch im nettest-Code keine Fehler gefunden wurden, bleibt als Ursache der stark abweichenden Messergebnisse nur das DuT. Um diese Annahme zu prüfen, wurden der Durchsatz-Test für 1024 und 1518 Byte mit beiden Messsystemen wiederholt, dabei wurden für beide Rahmenlängen 29 Messungen mit nettest und 13 mit dem Spirent-System durchgeführt. Die aufbereiteten Ergebnisse sind in **Tabelle 12** aufgeführt und bestätigen die Vermutung, dass das DuT die Ursache der starken Unterschiede ist. Der Mittelwert $\overline{X}$ und die Standardabweichung σ liegen für die Rahmenlänge 1024 sehr nahe beieinander. Dies lässt den Schluss zu, dass beide Messungen eine ähnliche Verteilung aufweisen. Für die Länge von 1518 Byte sind die Abweichungen von $\overline{X}$ und σ größer, jedoch ist auch hier eine Schwankung der Messewerte beider Messsysteme erkennbar. Da mit beiden Systeme eine ähnliche Verteilung der Messwerte ermittelt wurde, ist sicher, dass dieses DuT nicht als statisches System betrachtet werden kann. Um trotzdem eine repräsentative Messung durchzuführen, sollte die Anzahl der Wiederholungen erhöht werden, die konfigurierten drei Durchgänge sind scheinbar nicht ausreichend, um einen stabilen Mittelwert zu ermitteln. Alternativ ist auch eine Erhöhung der Traildauer möglich.

Tabelle 11: Messergebnisse: Durchsatz - DuT 1 Linux

L / Byte	$\mathbf{D_{nettest}}$ **/ MBit/s**	$\mathbf{D_{spirent}}$ **/ MBit/s**	Δ**D / MBit/s**	δ**D / % D$_{maxL2}$**
64	179,15	185,39	6,24	0,04
128	360,76	368,27	7,51	0,04
256	716,45	735,38	18,93	0,10
512	1433,60	1478,31	44,71	0,23
1024	3048,11	3478,93	430,81	2,20
1280	3648,88	3668,90	20,02	0,10
1420	3735,93	3748,55	12,62	0,06
1518	3562,90	3700,81	137,92	0,70

Tabelle 12: Auswertung: Durchsatz Test(1024,1518), Mittelwert $\overline{\mathbf{X}}$ und Standardabweichung σ, DuT 1

Messgerät, L / Byte	$\overline{\mathbf{X}}$ **/ MBit/s**	σ **/ MBit/s**	**max(D) − min(D)** / MBit/s
nettest, 1024	3337,38	171,31	509,34
Spirent, 1024	3327,61	167,46	416,08
nettest, 1518	3671,87	60,56	314,14
Spirent, 1518	3692,52	17,84	69,53

6.3.2 Auswertung der Latenz Tests

Die Laufzeit eines Ethernet Rahmens hängt unter anderem von dessen Länge ab, daher ist ein Anstieg der Laufzeiten für steigende Rahmenlängen zu erwarten. Die Latenz der vermessenen DuT hängt stark von deren Auslastung ab, je höher die Last um so höher auch die Latenz. In **Tabelle 13** sind die Messewerte für DuT 1 aufgeführt, diese entsprechen weitgehend den Erwartungen, für die Länge 1518 Byte ist jedoch mit nettest ein auffällig kleiner Wert gemessen wurden.

Tabelle 13: Messergebnisse: Latenz - DuT 1

L / Byte	64	128	256	512	1024	1280	1420	1518
$\mathbf{\Delta t_{nettest}}$ **/ ns**	173533	176139	179823	196945	245225	1688934	1754292	279688
$\mathbf{\Delta t_{spirent}}$ **/ ns**	161406	148969	156404	164592	266210	900224	1051820	2747702

In **Tabelle 14** sind die Messwerte beider Messsysteme sowie deren Differenz für DuT 1 aufgelistet. Der Anstieg der Latenz mit steigender Rahmenlänge entspricht dem erwarteten Ergebnis. Einzig die Messung mit nettest für 1518 Byte Rahmen entspricht mit 260 μs nicht der Erwartung. Da nettest die Ergebnisse des Durchsatz-Tests als Lastvorgabewerte für den Latenztest verwendet, wirken sich die zuvor

beschriebenen Abweichungen auch auf diesen Test aus. Es ist bekannt, dass die Latenz des DuT bei gleicher Rahmenlänge mit steigender Last zunimmt. Daher ist es wahrscheinlich, dass der zu gering erscheinende Messwert durch die um 137 MBit/s geringere Datenrate verursacht wurde. Für Rahmenlängen ab 1024 Byte sind die Abweichungen der Messwerte zwischen den Messsystemen sehr groß. Die unterschiedlichen Datenraten, bei der die Latenz gemessen wurde und die variierende Leistungsfähigkeit des DuT sind mögliche Ursachen für die starken Abweichung.

Weiterhin unterscheidet sich das Messverfahren zwischen nettest und dem Spirent Messsystem. Während in nettest Latenz- und Durchsatz-Test nacheinander ausgeführt werden misst das Spirent Testcenter die Latenz direkt während des Durchsatz-Tests. Daher wirkt sich die schwankende Leistungsfähigkeit des DuT weniger stark auf den Latenz-Tests aus. Weiterhin misst das Spirent System die Latenz für jeden Rahmen, während nettest nur mit einer Abtastrate von maximal 3 kHz misst. Ausgehend von den Differenzen der Messwerte für die Rahmenlängen von 64 bis 512 Byte ist eine relativ geringe Abweichung beider Messsysteme zu beobachten, was zu den ebenfalls geringen Abweichungen der gemessenen Datenraten passt. Grundsätzlich sind die einzelnen Latenzmessungen ,welche nettest durchführt, sehr genau. Im Paper [EMM+2015] wird die Genauigkeit und Präzision des verwendeten Latenzmessverfahrens untersucht. Es wird eine Genauigkeit von 19,2 ns ermittelt, mit einer Präzision von ±6,4 ns.

Tabelle 14: Auswertung: Latenz Test, Differenz nettest zu Spirent, DuT 1

L / MBit/s	64	128	256	512	1024	1280	1420	1518
$\mathbf{t_{spirent} - t_{nettest}}$ / μs	-12	-27	-23	-32	21	-789	-703	2468
$\mathbf{t_{spirent}}$ / μs	161	149	156	165	266	900	1052	2748
$\mathbf{t_{nettest}}$ / μs	174	176	180	197	245	1689	1754	280

Auswertung des Frameloss Tests

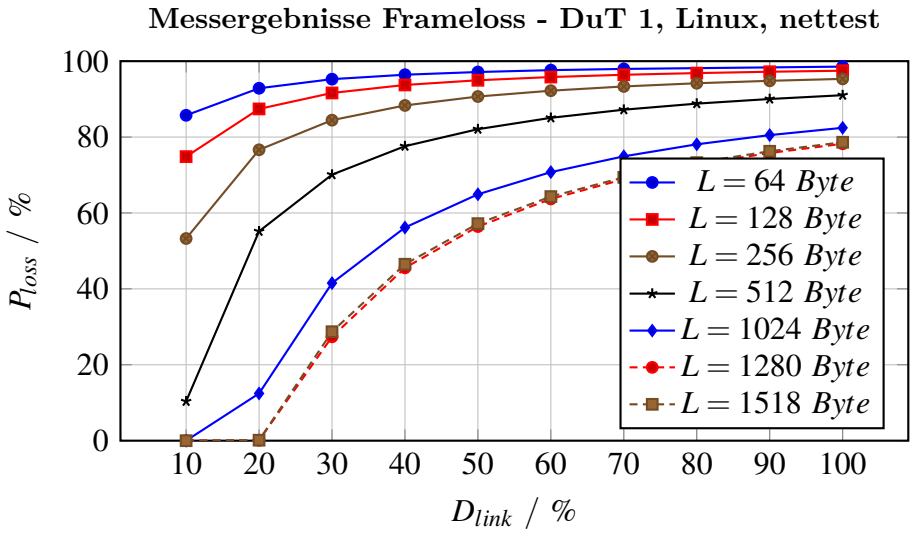


Abbildung 19: Messergebnisse: Frameloss - DuT 1, Linux, nettest

Die Frameloss Messung sollte mit sinkender Datenrate einen geringeren Prozentsatz verlorener Pakete zeigen. Weiterhin ist zu erwarten, dass bei steigender Rahmenlänge und gleicher Datenrate eine geringere Paketverlustrate gemessen wird. Aufgrund der Umfangreichen Messwerte wird in **Abbildung 19** nur der Verlauf für das Messsystem nettest dargestellt. Die Werte entsprechen den Erwartungen und sind plausibel. Um die umfangreichen Messwerte auszuwerten ist in **Abbildung 20** die Differenz der Paketverlustraten $\Delta Frameloss$[6] aller Rahmenlängen über der prozentualen Datenrate aufgetragen. Für 1024, 1280 sowie 1518 wird bei 10 % der maximalen Datenrate kein Paketverlust gemessen, daher ist die Differenz Null. Es ist deutlich erkennbar, dass die Abweichungen der Messungen mit steigender Datenrate geringer werden. Da der relative Anteil der weitergeleitete Rahmen mit steigender Datenrate geringer wird, ist auch der Einfluss, den das DuT auf die Abweichungen der Messungen hat, kleiner. Daher kann angenommen werden, dass die Genauigkeit der Frameloss-Messung unter 1 % liegt.

[6] $\Delta Frameloss = Frameloss_{spirent} - Frameloss_{nettest}$

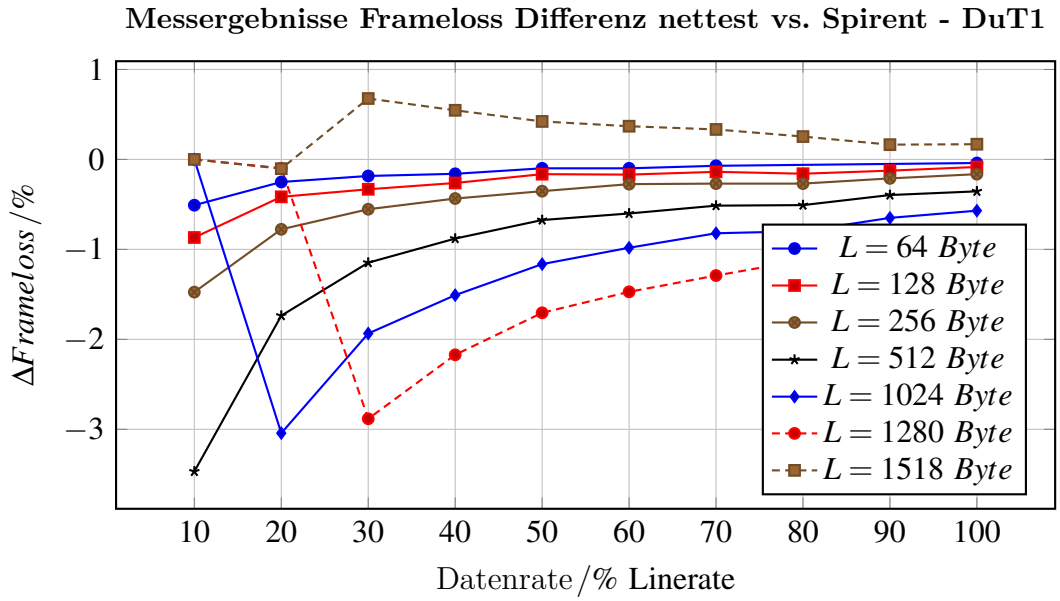


Abbildung 20: Frameloss Messung DuT1: Abweichung nettest vs. Spirent

6.3.3 Auswertung der BackToBack-Tests

Die Messwerte des BackToBack-Tests stehen im Zusammenhang mit denen des Durchsatztests für die jeweilige Rahmenlänge. Falls das DuT für eine Rahmenlänge die volle Geschwindigkeit der Schnittstelle erreicht (Linerate), so ist die Burstlänge N theoretisch unendlich, und wird durch die vorgegebene Obergrenze der Testkonfiguration beschränkt. Diese wird im EXFO Messsystem durch die maximale Laufzeit festgelegt und ist daher abhängig von der Rahmenlänge, nettest und Spirent verwenden ein fixe Anzahl. **Tabelle 15** gibt die Messwerte für DuT 2 mit pfSense im Routerbetrieb wieder. Es ist offensichtlich, dass ab einer Rahmenlänge 1024 Byte Linerate erreicht wird.

Die in **Tabelle 16** aufgeführten Ergebnisse des BackToBack-Tests beider Messsysteme stimmen sehr gut überein. Die maximal erreichbare Datenrate des DuT sowie die beobachteten Schwankungen haben nur einen geringen Einfluss auf die Ergebnisse des BackToBack-Tests.

Tabelle 15: Messergebnisse: Back to Back - DuT 2, pfSense

L / Byte	$N_{nettest}$/Rahmen	N_{EXFO}/Rahmen
64	1145	1320
128	1259	1742
256	1641	4510
512	3987	11477
1024	100000	598660
1280	100000	480770
1420	100000	nM
1518	100000	406372

Tabelle 16: Messergebnisse: Back to Back - DuT 1

Rahmenlänge / Byte	Burstlänge *nettest*	Burstlänge *Spirent*	Δ
64	481	489	8
128	481	489	8
256	481	508	27
512	505	527	22
1024	531	547	16
1280	547	586	39
1420	554	586	32
1518	562	586	24

6.3.4 Auswertung der IMIX-Tests

Die Messergebnisse der IMIX-Messung sollten in etwa dem Durchsatz-Wert entsprechen, der mit dem RFC2544 Durchsatz-Test für die durchschnittliche Paketgröße ermittelt werden würde. Die mittlere Rahmenlänge entspricht für den verwendeten IMIX 417,38 Byte. Durch lineare Approximation aus den Ergebnissen der RFC2544 Durchsatzmessung bei 256 und 512 Byte lässt sich der Erwartungswert von etwa 1 169[7] errechnen.

Tabelle 17: Messergebnisse: IMIX DuT 1

IMIX Paketverteilung	$D_{nettest}$	$D_{Spirent}$
Spirent IPsec	371,46	1 189,11

[7] $1169\ MBit/s \approx \frac{1\,433{,}60\ MBit/s - 716{,}45\ MBit/s}{512\ Byte - 256\ Byte} * 417{,}38\ Byte$

Die Messwerte des IMIX-Tests unterscheiden sich erheblich zwischen nettest und Spirent. Mit nettest wird etwa um den Faktor 3.2 weniger Durchsatz gemessen. Dafür ist sehr wahrscheinlich die Reihenfolge der Rahmen im Schedule verantwortlich: Für nettest entsteht aufgrund der hohen Anzahl von 90 Byte Rahmen (siehe Anhang C - Tabelle 20) ein Schedule, welcher, nachdem alle anderen Längen aufgebraucht sind, ausschließlich mit kleine Rahmen gefüllt wird. Dadurch wird das DuT für längere Zeit ausschließlich mit kurzen Rahmen belastet. Weil die Datenrate durch den TRS konstant gehalten wird steigt die Paketrate an. Wenn dieser Abschnitt des Schedules zu groß ist, werden Pakete im DuT verworfen. Das hat zur Folge, dass der Suchalgorithmus die Datenrate absenkt. Das Spirent Testcenter wiest für die gleichen IMIX-Parameter eine gleichmäßigere Verteilung auf und misst daher vermutlich höhere Datenraten.

7 Zusammenfassung und Ausblick

Zusammenfassung der Ergebnisse

Es konnte gezeigt werden, dass es mit Standard x86 Rechnersystemen und handelsüblichen Netzwerkschnittellen möglich ist, ein Messsystem für 10GBit/s Ethernet Komponenten zu entwickeln, welches auch für kleinste Rahmenlänge die volle Datenrate erreicht und dabei verlorene sowie umgeordnete Rahmen erkennen kann. Außerdem konnte durch Messungen nachgewiesen werden, dass auch eine CPU mit zehn Kernen ausreichend Rechenleistung bietet, um zwei Netzwerkschnittstellen bei maximaler Paketrate zu bedienen. Ein Zwei-Sockel-System ist daher nicht nötig. Dabei ist jedoch zu beachten, dass die X520-T2 Dual-Port Netzwerkkarte im Vollduplex-Betrieb einen Engpass darstellt, wenn mit 64 Byte Rahmenlänge gemessen wird. Daher sollten Netzwerkkarten mit nur einer Schnittstelle verwendet werden.

Die Messergebnisse sind für drei Tests des RFC2544 Standards sowie die des IMIX Durchsatz-Tests plausibel, einzig der Latenztest ergab teilweise unglaubwürdige Ergebnisse. Es ist nur zum Teil möglich die durchgeführten Messungen direkt miteinander zu verglichen, da das DuT für erhebliche Schwankungen der Durchsatzmesswerte verantwortlich ist. Es konnte jedoch durch statistische Auswertung weiterer Messungen nachgewiesen werden, dass die mit nettest ermittelte Verteilung der Durchsatzwerte sehr ähnlich der des Spirent Messsystems ist. Für die Genauigkeit der Datenraten und den möglichen Messbereich ist die ordnungsgemäße Funktion des TRS entscheidend. Hier wurden bei den verwendeten Netzwerkchips Einschränkungen gefunden und dokumentiert. Der Messbereich des entwickelten Messsystems wird durch diese begrenzt. Es ist für Rahmenlänge ab 80 Byte aufwärts möglich, Datenrate von 10 MBit/s bis 10000 MBit/s zu messen. Für die Rahmenlängen von 64 bis 79 Byte ist es zwar möglich mit maximaler Datenrate Rahmen senden und empfangen zu können, jedoch ist ein Drosseln der Datenrate nur im Bereich 10 bis 8500MBit/s möglich. Aufgrund von Hardwareeinschränkungen der Netzwerkchips wird für höhere eingestellte Datenraten nur mit erheblich geringeren Raten gesendet. Somit ist in diesem Bereich keine Messung möglich. Wie in Abbildung 5 ersichtlich ist, wird theoretisch eine minimale Genauigkeit der Sendedatenrate unter 1,2 % der maximalen Ethernet Datenrate erreicht. Praktisch wird eine bessere Genauigkeit

erreicht, weil erst ab 8500 MBit/s für kleine Rahmenlängen gemessen werden kann. Somit kann für den nutzbaren Messbereich der Datenraten mit einer Genauigkeit von unter 0.8 % angegeben werden. Weiterhin ist mit steigender Rahmenlänge und sinkender Datenrate eine erheblich höher Genauigkeit möglich.

Um die Latenzmessung auch für maximale Paketraten möglich zu machen war es nötig, dass die diese mit einem zweiten Datenstrom, parallel zu dem Laststrom, durchgeführt werden. Dadurch wird die Latenz nicht für jedes Datenpaket ermittelt. Der RFC2544 Standard sieht eine getrennte Messung der maximalen Datenrate und durchschnittlichen Latenz vor. Es hat sich gezeigt, dass dieses Vorgehen für DuT mit variierender Leistungsfähigkeit einen großen Nachteil birgt. Weil die Latenzmessung nicht zum gleichen Zeitpunkt erfolgt, kann nicht davon ausgegangen werden, dass die Leistung des DuT der der Durchsatzmessung entspricht. Daher sind die Messwerte des Latenztests nur für relativ stationäre DuT, bezogen auf die Leistungsfähigkeit, brauchbar.

Anhand des entwickelten IMIX-Tests konnten die Nachteile eines nicht standardisierten Tests nachvollzogen werden. Durch andere Anordnung der Rahmenlängen im Datenstrom wurde eine um Faktor 3,2 abweichende Datenrate gegenüber dem Spirent Messsystem gemessen.

Insgesamt konnte gezeigt werden, dass es möglich ist, ein Messsystem für 10GBit/s Netzwerkkomponenten auf OSI-Schicht 3, basierend auf Standard Hardware, zu entwickeln. Es hat sich jedoch besonders bei der Latenzmessung herausgestellt, dass hier noch Potenzial für Verbesserungen besteht. Nettest ist den Funktionsumfang betreffend noch nicht mit dem des Spirent Testcenters vergleichbar. Durch Verwendung von quelloffener Software und einer skalierbaren Architektur ist die Erweiterung und Anpassung an die Bedürfnisse des Anwenders einfacher und flexibler als bei kommerziellen Systemen. Ein wichtiger Pluspunkt für nettest ist, neben den funktionalen Aspekten, der Kostenfaktor: Für ein Zwanzigstel der Anschaffungskosten kann ein Messsystem realisiert werden, das trotz der genannten Einschränkungen für viele Anwendungen einsetzbar ist. Besonders im Bereich softwarebasierte VPN Gateways und Router wird für kleine Rahmenlängen oft nur ein Bruchteil der theoretischen Datenrate erreicht(siehe Datenblatt [SEC2016]). Daher sind die Einschränkungen des Messbereichs für diese Art DuT nicht relevant.

Ausblick

Die Genauigkeit des Messsystems für den Durchsatz-, Frameloss-, und IMIX-Test hängt maßgeblich von der Möglichkeit ab, wie genau die Sendedatenrate eingestellt werden kann. Weil für geringere Datenraten eine höhere Auflösung erzielt wird, könnte man den Lastdatenstrom auf mehrere Sende-Queues aufteilen. Somit würde gleichzeitig die Einschränkung für hohe Datenrate bei kurzen Rahmenlängen umgangen werden. Auch die Simulation mehrerer Datenströme, die sich durch IP-Adresse oder Port unterscheiden, ist eine Funktion, die viele kommerzielle Messsysteme bieten. Da die Datenverarbeitung in Netzwerkkomponenten vermehrt parallel erfolgt und die Verteilung auf die Recheneinheiten meist anhand der Header der OSI-Schichten 3 und 4 durchgeführt wird, wäre es so möglich, die Skalierbarkeit dieser Systeme zu vermessen.

Eine Weiterentwicklung des Latenzmessverfahrens durch die Integration in den Durchsatztest würde die Genauigkeit der Messwerte vermutlich verbessern. Dafür wäre es sinnvoll, zu untersuchen, wie stark sich das Einbringen und Auswerten eines Zeitstempels für jeden Rahmen auf die Leistungsfähigkeit auswirkt. In Verbindung mit mehreren Datenströmen könnte es möglich sein, für jedes Pakete eine Latenzmessung durchzuführen.

Quellenverzeichnis

[BM1999] Bradner, S. und J. McQuaid (1999). *Benchmarking Methodology for Network Interconnect Devices*. URL: https://rfc-editor.org/rfc/rfc2544.txt.

[DEC] DE-CIX (2016). *DE-CIX - Statistics*. URL: https://www.de-cix.net/about/statistics/ (besucht am 25. 07. 2016).

[EMM+2015] Emmerich, Paul u. a. (2015). „MoonGen: A Scriptable High-Speed Packet Generator“. In: *Internet Measurement Conference (IMC) 2015*. Tokyo, Japan.

[GNU] GNULinux (2016). *GNU/Linux Manpages: time - overview of time and timers*. URL: http://manpages.ubuntu.com/manpages/xenial/en/man7/time.7.html (besucht am 10. 07. 2016).

[HER2006] Herck, Tim Van (2006). URL: http://www.ietf.org/mail-archive/web/bmwg/current/msg00691.html (besucht am 17. 08. 2016).

[INT2012] Intel (2012). *Product brief, Intel® Ethernet Converged Network Adapter X520*. Intel. URL: http://www.intel.com/content/dam/doc/product-brief/ethernet-x520-server-adapters-brief.pdf (besucht am 17. 08. 2016).

[INT2015] — (2015). *Intel® Xeon® Processor E5 and E7 v3 Family Uncore Performance Monitoring Reference Manual*. Intel. URL: http://www.intel.com/content/dam/www/public/us/en/zip/xeon-e5-v3-uncore-performance-monitoring.zip (besucht am 17. 08. 2016).

[INT2016] — (2016). *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B Part2*. URL: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf (besucht am 18. 08. 2016).

[JT2007] John, Wolfgang und Sven Tafvelin (2007). „Analysis of Internet Backbone Traffic and Header Anomalies Observed“. In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. IMC '07. San Diego, California, USA: ACM, S. 111–116. ISBN: 978-1-59593-908-1. DOI: 10.1145/1298306.1298321. URL: http://conferences.sigcomm.org/imc/2007/papers/imc91.pdf.

[KAR2013] Karedla, Rama (2013). *Intel Xeon E5-2600 v3 (Haswell) Architecture & Features*. Intel. URL: http://repnop.org/pd/slides/PD_Haswell_Architecture.pdf (besucht am 17. 08. 2016).

[MOL+2015] Molka, D. u. a. (2015). „Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture“. In: *Parallel Processing (ICPP), 2015 44th International Conference on*, S. 739–748. DOI: 10.1109/ICPP.2015.83.

[MOR1999] Morton, Al (1999). *IMIX Genome: Specification of Variable Packet Sizes for Additional Testing*. URL: https://tools.ietf.org/html/rfc6985.

[PCI2016] PCISIG (2016). *PCI-SIG: Frequently Asked Questions*. URL: https://pcisig.com/faq?&keys=&page=2 (besucht am 18. 08. 2016).

[RIZ2012] Rizzo, Luigi (2012). *netmap: a novel framework for fast packet I/O*. Universita di Pisa, Italy. URL: http://info.iet.unipi.it/~luigi/papers/20120503-netmap-atc12.pdf (besucht am 05. 06. 2016).

[SEC2016] secunet (2016). *Technisches Datenblatt: VPN-Gateway für VS-NfD-konforme Kommunikation*. secunet Security Networks AG. URL: https://www.secunet.com/fileadmin//user_upload/Download/Printmaterial/Factsheets/deutsch/sn_SINA_L3_Box_S_FS_D.pdf (besucht am 29. 09. 2016).

[SPI] Spirent (2016). *Spirent fX2 datasheet*. URL: https://www.spirent.com/-/media/Datasheets/Broadband/PAB/SpirentTestCenter/Spirent_fX2_datasheet.pdf?la=en (besucht am 25. 09. 2016).

[TAN2003] Tanenbaum, Andrew S. (2003). *Computernetzwerke*. 4., überarb. A. Pearson Studium. Pearson Studium. ISBN: 9783827370464.

Selbstständigkeitserklärung

Der Verfasser erklärt, dass er die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Dresden, 20. Januar 2017

Danksagungen

An dieser Stelle möchte ich all jenen danken, die durch ihre fachliche und persönliche Unterstützung zum Gelingen dieser Diplomarbeit beigetragen haben.

Mein besondere Dank gilt Herrn Prof. Dr.-Ing Vogt von der HTW Dresden für die freundliche und unkomplizierte Betreuung meiner Abschlussarbeit. Ebenso danke ich Dipl.-Ing. Robert Menzel für die geduldige Betreuung sowie der secunet Security Networks AG, die meine Diplomarbeit sowohl durch fachliche Unterstützung als auch durch die Bereitstellung der nötigen Hardware und Messsysteme, erst ermöglicht haben.

Mein besonderer Dank gilt meiner Familie, insbesondere meinen Eltern, die mir mein Studium ermöglicht und mich in all meinen Entscheidungen unterstützt haben.

Zu guter Letzt sei den vielen Autoren hilfreicher Werkzeuge wie LaTeX, Gnuplot und KOMA-Script gedankt.

A Bewertungsmatrix für netmap, DPDK und MoonGen

Tabelle 18: Bertungsmatrix für netmap, DPDK und MoonGen

Name	netmap API	DPDK	MoonGen
Lizenz	BSD	BSD, Linux Treiber: GPL2	MIT License
Betriebssystem	FreeBSD	Linux	Linux
Hardware Offloading	nein	ja	ja
Direkter Register Zugriff	nein	ja	ja
Optimierungsgrad	+	++	++
Community Größe	-	++	-
Dokumentation	++	++	mangelhaft
Protokoll Header Unterstützung	keine	keine	Ethernet,IPv4, UDP, TCP, ARP, DNS, ICMP,..
Programmiersprache	C	C	Lua , C
Netzwerkchip Unterstützung	+	++	gering
HW Zeitstempel	nein	nein	ja
Interface Statistik	nein	ja	ja
Queue Statistik	nein	ja	nein

B Hardware Komponenten der DuT

Tabelle 19: Hardware der DuT Systeme

Komponente	**DuT 1**	**DuT 2**
CPU	Intel Xeon E5-2680v3 1x12@2.7GHz	2x Intel Xeon E5645 2x6x@2.4GHz
NIC	Silicom PE310G4I71L 4x10G	Intel i350 1G
RAN	8GB DDR4	4GB DDR3

C IMIX Parameter

Tabelle 20: Testparameter: IMIX Zusammensetzung

Länge / Byte	**Anzahl / Pakete**	**Anteil Pakete / %**	**Anteil L1 Datenrate / %**
90	5867	58,67	14,76
92	200	2,00	0,51
594	2366	23,66	33,21
1418	1567	15,67	51,52

D Messergebnisse

Tabelle 21: Messergebnisse: Durchsatz, DuT 1, Linux

L / Byte	**$D_{nettest}$ / MBit/s**	**$D_{spirent}$ / MBit/s**
64	179,15	185,39
128	360,76	368,27
256	716,45	735,38
512	1433,60	1478,31
1024	3048,11	3478,93
1280	3648,88	3668,90
1420	3735,93	3748,55
1518	3562,90	3700,81

Tabelle 22: Messergebnisse: Latenz, DuT 1, Linux

L / Byte	64	128	256	512	1024	1280	1420	1518
$\Delta t_{nettest}$ / ns	173533	176139	179823	196945	245225	1688934	1754292	279688
$\Delta t_{spirent}$ / ns	161406	148969	156404	164592	266210	900224	1051820	2747702

Tabelle 23: Messergebnisse: BackToBack, DuT 1, Linux

L / Byte	**$N_{nettest}$ / Rahmen**	**$N_{spirent}$ / Rahmen**	**$N_{spirent} - N_{nettest}$ / Rahmen**
64	481	489	8
128	481	489	8
256	481	508	27
512	505	527	22
1024	531	547	16
1280	547	586	39
1420	554	586	32
1518	562	586	24

Tabelle 24: Auswertung: DuT 1, Linux, Frameloss Differenz $P_{loss,Spirent} - P_{loss,nettest}$

Load / %	**64**	**128**	**256**	**512**	**1024**	**1280**	**1518**
10	-0,509	-0,868	-1,475	-3,469	0	0	0
20	-0,251	-0,416	-0,777	-1,737	-3,043	-0,098	-0,103
30	-0,184	-0,332	-0,554	-1,148	-1,935	-2,883	0,677
40	-0,160	-0,262	-0,435	-0,880	-1,509	-2,172	0,546
50	-0,099	-0,164	-0,353	-0,674	-1,164	-1,705	0,422
60	-0,099	-0,169	-0,275	-0,601	-0,982	-1,472	0,369
70	-0,071	-0,138	-0,269	-0,514	-0,821	-1,291	0,333
80	-	-0,159	-0,269	-0,508	-0,798	-1,121	0,254
90	-	-0,125	-0,211	-0,396	-0,650	-0,987	0,164
100	-0,040	-0,084	-0,164	-0,355	-0,570	-0,868	0,169

Tabelle 25: Messergebnisse: IMIX, DuT 1, Linux

IMIX Paketverteilung	**$D_{nettest}$ / MBit/s**	**$D_{Spirent}$ / MBit/s**
Spirent IPsec	371,46	1 189,11

Tabelle 26: Messergebnisse: Durchsatz, DuT 1, Linux - $L = \{1024, 1518\}$

$\mathbf{D_{nettest}}$ / MBit/s		$\mathbf{D_{spirent}}$ / MBit/s	
1024	1518	1024	1518
3457,03	3751,34	3.030,52	3.639,41
3455,03	3751,34	3.030,52	3.685,09
3450,59	3694,15	3.065,00	3.685,50
3448,59	3694,15	3.306,38	3.687,12
3448,59	3690,41	3.375,34	3.690,78
3442,51	3690,41	3.404,98	3.694,71
3442,51	3690,41	3.409,83	3.698,10
3442,38	3690,41	3.418,45	3.698,37
3442,19	3689,41	3.437,98	3.699,73
3442,19	3688,96	3.444,44	3.704,61
3442,15	3688,46	3.444,44	3.704,61
3441,16	3687,74	3.444,44	3.705,83
3441,16	3687,74	3.446,60	3.708,95
3440,82	3686,52		
3427,73	3682,52		
3417,46	3682,08		
3417,46	3682,08		
3417,05	3679,64		
3388,37	3677,41		
3388,37	3676,76		
3376,46	3671,57		
3280,94	3671,57		
3280,94	3671,57		
3112,79	3671,57		
3093,37	3667,91		
3092,05	3667,91		
2958,68	3665,88		
2947,69	3497,19		
2947,69	3437,19		

Tabelle 27: Messergebnisse: Durchsatz, DuT 2, Linux

L / Byte	**$D_{nettest}$ / MBit/s**	**D_{EXFO} / MBit/s**	**ΔD / MBit/s**	**Linerate L2 / %**
64	142,33	143,61	1,28	0,08
128	275,26	278,77	3,50	0,20
256	513,01	528,38	15,37	0,83
512	1052,04	1073,76	21,71	1,13
1024	1829,69	1840,07	10,38	0,53
1280	1868,56	1868,61	0,05	0,00
1518	1540,93	1812,54	271,61	13,76

Tabelle 28: Messergebnisse: Latenz, DuT 2, Linux

L / Byte	64	128	256	512	1024	1280	1420	1518
$\Delta t_{nettest}$ / ns	165031	-	180916	296192	293192	633115	646811	427431
Δt_{exfo} / ns	145978	170491	187047	964616	499679	689464	-	642241

Tabelle 29: Messergebnisse: BackToBack, DuT 2, Linux

L / Byte	**$N_{nettest}$/Rahmen**	**N_{EXFO}/Rahmen**	**$N_{EXFO} - N_{nettest}$ / Rahmen**
64	775	785	10
128	566	588	22
256	483	530	47
512	620	776	156
1024	16191	16214	23
1280	20057	20059	2
1420	20878		-
1518	6226	6219	-7

Tabelle 30: Messergebnisse: IMIX, DuT 2, Linux

IMIX Paketverteilung	**$D_{nettest}$**	**D_{EXFO}**
Spirent IPsec	267,32	-

Tabelle 31: Messergebnisse: Durchsatz, DuT 2, pfSense

L / Byte	**$D_{nettest}$ / MBit/s**	**D_{EXFO} / MBit/s**	**ΔD / MBit/s**	**Linerate L2 / %**
64	90,18	95,81	-5,63	-0,74
128	180,36	191,33	-10,97	-1,27
256	334,82	382,66	-47,84	-5,16
512	718,42	765,70	-47,28	-4,91
1024	980,81	980,84	-0,03	0,00
1280	984,59	984,62	-0,03	0,00
1518	986,97	987,00	-0,03	0,00

Tabelle 32: Messergebnisse: Latenz, DuT 2, pfSense

L / Byte	64	128	256	512	1024	1280	1420	1518
$\Delta t_{nettest}$ / ns	149206	852218	45872	512863	305429	311357	319843	317118
Δt_{exfo} / ns	5575697	5571867	5603984	5601665	298264	301167	-	268270

Tabelle 33: Messergebnisse: BackToBack, DuT 2, pfSense

L / Byte	**$N_{nettest}$/Rahmen**	**N_{EXFO}/Rahmen**	**$N_{EXFO} - N_{nettest}$ / Rahmen**
64	1145	1320	175
128	1259	1742	483
256	1641	4510	2869
512	3987	11477	7490
1024	100000	598660	0(Sättigung)
1280	100000	480770	0(Sättigung)
1420	100000	nM	-
1518	100000	406372	0(Sättigung)

Tabelle 34: Messergebnisse: IMIX, DuT 2, pfSense

IMIX Paketverteilung	**$D_{nettest}$**	**D_{EXFO}**
Spirent IPsec	469,01	-