Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InES Institute of
Embedded Systems

Bachelorarbeit Informatik
BA15 wlan 1

Ermittlung der Performance von Netzwerkfunktionen am Beispiel von PRP

Autoren	Mauro Guadagnini (guadamau@students.zhaw.ch) Prosper Leibundgut (leibupro@students.zhaw.ch)
Hauptbetreuung	Prof. Hans Weibel (wlan@zhaw.ch)
Nebenbetreuung	David Blatter (blad@zhaw.ch)
Datum	05.06.2015

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:

……

Unterschriften:

……...

……

Zusammenfassung

Lorem ipsum

Abstract

Lorem ipsum.

Vorwort

Arbeit «Ermittlung der Performance von Netzwerkfunktionen am Beispiel von PRP»

Inhaltsverzeichnis

Teil I

Einführung und Grundlagen

1 Einleitung

1.1 Ausgangslage

1.1.1 Stand der Technik

Der Standard zu PRP Version 1 (oder PRP-1) wurde unter dem Namen IEC 62439-3 am 05.07.2012 veröffentlicht, welcher eine technische Revision des ursprünglichen Standards (PRP Version 0 oder PRP-0) vom 25.02.2010 darstellt und diesen für ungültig erklärt. PRP-1 wurde unter anderem entwickelt, damit es mit einem weiteren Protokoll für redundante Netzwerkkommunikation, HSR (High-availability Seamless Redundancy) kompatibel ist, jedoch ging dabei die Kompatibilität zu PRP-0 verloren. [54]

PRP ist bereits bei einigen Firmen implementiert und wird unter anderem für Substation Automation verwendet. [4, 25]

In dieser Arbeit, welche die Performance-Ermittlung von Netzwerkfunktionen umfasst, wird lediglich PRP Version 1 behandelt.

1.1.2 Bestehende Arbeiten

Zu PRP-1 existieren Arbeiten, welche sich mit der Verwendung von PRP-1 in Substation-Automation-Systemen auseinander setzen [4] oder Verbesserungen vorschlagen, welche in einem eventuell zukünftigen PRP-2 implementiert werden könnten [43].

Eine Arbeit, welche sich konkret mit der Performance-Ermittlung von Netzwerkfunktionen am Beispiel von PRP auseinandersetzt, konnte nicht aufgefunden werden.

1.2 Zielsetzung / Aufgabenstellung / Anforderungen

1.2.1 Anforderungen

Durch das Institute of Embedded Systems der ZHAW wurde den Autoren am 10. Februar 2015 eine Aufgabenstellung [22] (siehe Kapitel 11 auf Seite 159) zugestellt, welche die nachfolgenden Hauptanforderungen umfasst:

- Studium von PRP und Aufbau Testumgebung
- Ermittlung der Leistungsfähigkeit der Implementierung unter manigfaltigen Aspekten
- Untersuchung, Bewertung und ggf. Entwicklung von Tools
- Einfluss von Offload-Mechanismen aufzeigen
- Auswirkung von PRP auf das Endsystem, insbesondere auf die Belastung von CPU und Memory studieren
- Grundsätze und Rahmenbedingungen formulieren

1.2.2 Erwartetes Resultat

Lorem ipsum

1.2.3 Vorausgesetztes Wissen

In den theoretischen Grundlagen (siehe Kapitel 2 auf der nächsten Seite) werden unter anderem das PRP-Protokoll und dessen Software-Implementation behandelt.

Zum Verständnis dieser Projektarbeit ist ein Vorwissen über die allgemeine Netzwerkkommunikation nötig. Dieses Vorwissen umfasst folgende Bereiche:

- Allgemeine Netzwerk- und Hardware-Begriffe wie z.B. MAC-Adresse, Ethernet-Port oder Ethernet-Frame.
- Funktionsweise eines Netzwerks inklusive der Übertragung eines Ethernet-Frames und dem Aufbau dessen Headers.

2 Theoretische Grundlagen

2.1 Parallel Redundancy Protocol (PRP)

Bei PRP handelt es sich um ein Kommunikationsprotokoll auf Layer 2, welches eine Redundanz im Netzwerk gewährleistet und in den Knoten statt dem Netzwerk implementiert ist. Dies wird erreicht, indem ein Netzwerkknoten mit 2 Netzwerkinterfaces an zwei disjunkten, parallel betriebenen LANs («LAN_A» und «LAN_B») angeschlossen wird, die unabhängig von einander sind. Diese beiden LANs, die sich bezüglich Performance und Topologie unterscheiden können, ergeben zusammen ein PRP-Netzwerk. Ein solcher Knoten im PRP-Netzwerk wird «Double Attached Node» (oder auch u.a. «Dual Attached Node»), kurz DAN, genannt. Ein DAN hat dieselbe MAC-Adresse auf beiden Netzwerkinterfaces. Da es sich hier um DANs handelt, die PRP implementiert haben, werden sie auch mit DANP («Double Attached Node implementing PRP») bezeichnet. In dieser Arbeit sind DAN und DANP gleichbedeutend. [54]

Wenn ein DANP etwas an einen anderen Netzwerkteilnehmer sendet, dupliziert er das Frame und übermittelt es über beide LANs. Beim Empfänger wird das Frame, das als Erstes ankommt, an die oberen Schichten weitergereicht und das Duplikat je nach nach Methode akzeptiert oder verworfen. Wie Duplikate erkennt und entfernt werden ist der Implementation überlassen. Auf die Duplikat-Handhabung wird genauer im Kapitel 2.1.1.2 auf Seite 17 eingegangen. Für die Duplikaterzeugung und -erkennung ist die Link Redundancy Entitiy (LRE) zuständig, welche sich zwischen den beiden Netzwerkinterfaces und den oberen Netzwerkschichten in einem DAN befindet. Um diese Frame-Redundanz handhaben zu können, wird vom LRE beim Versand dem Frame am Ende ein Redundancy Control Trailer (RCT) angehängt, der beim Empfänger von dessen LRE wieder entfernt wird und anhand von dem RCT Duplikate erkannt werden können. [23]

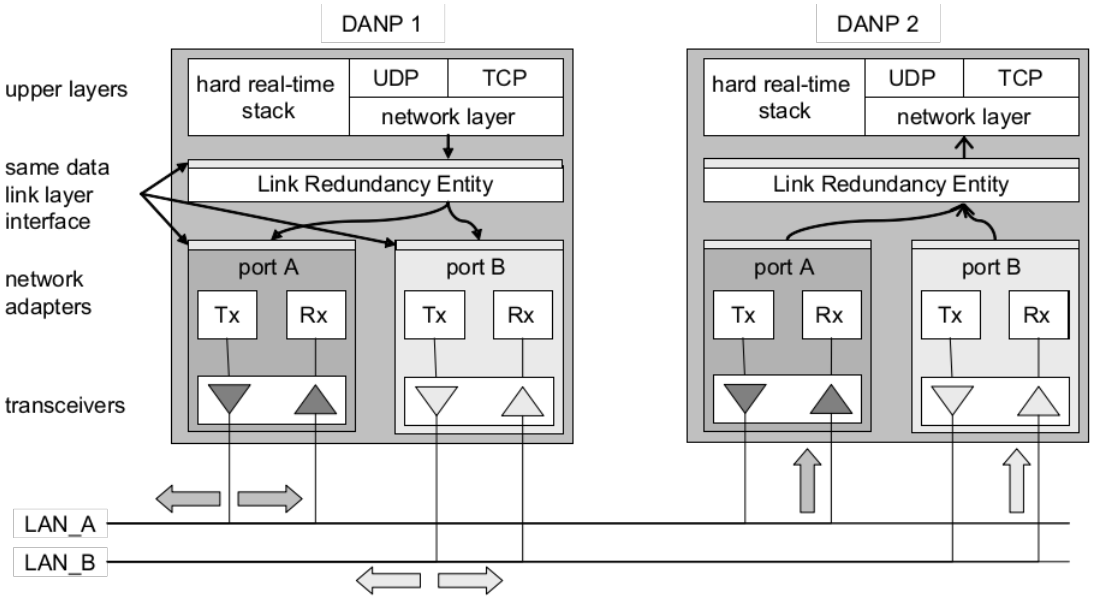


Abbildung 2.1: PRP-Netzwerk mit 2 kommunizierenden DANPs [54]

Dieser RCT hat eine Grösse von 6 Bytes und beeinhaltet folgende Parameter [54]:

- Sequenznummer (16 Bit)
  Jede Quelle hat lediglich einen Sequenznummernraum [28]

- LAN-Identifikator (4 Bit)
  Dieser Parameter lautet entwerder 0xA für LAN_A oder 0xB für LAN_B

- LSDU-Size (12 Bit)
  Umfasst die Grösse der LSDU (Link Service Data Unit) in Bytes zuzüglich der Grösse vom RCT

- PRP-Suffix (16 Bit)
  Damit ein Frame als PRP-1-Frame erkannt wird, wird der Suffix auf 0x88FB gesetzt [43]
  Die Länge des Suffix hat den Grund, dass so mit einer sehr geringen Wahrscheinlichkeit durch Zufall ein Nicht-PRP-Frame diesen Wert am Schluss des Frames (vor der Frame Check Sequence) hat und es als PRP-Frame erkannt wird, obwohl es keines wäre.

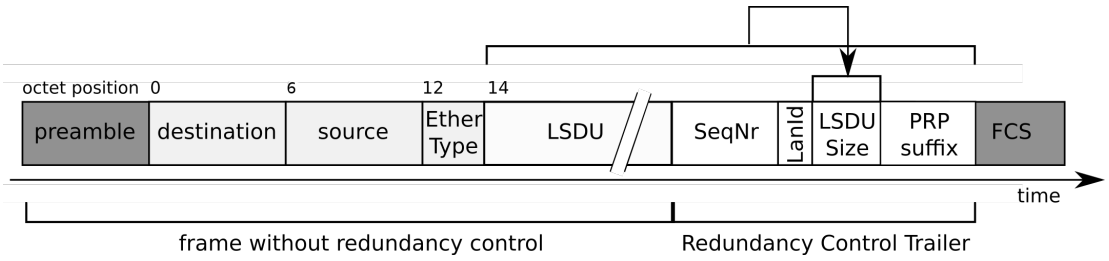


Abbildung 2.2: PRP-Frame mit RCT [54]

Des Weiteren gibt es auch Netzwerkkomponenten, die nur über ein Netzwerkinterface verfügen und lediglich an einem LAN des PRP-Netzwerks angeschlossen sind. Solche Geräte werden «Single Attached Node», kurz SAN, genannt. Diese können zwar mit allnen Netzwerkteilnehmern kommunizieren, die sich in beiden LANs befinden, jedoch können nur andere SANs erreicht werden, die am selben LAN angeschlossen sind. SANs, die mit dem anderen LAN verbunden sind, können nicht erreicht werden. Ein SAN weiss nichts von PRP. Daher erzeugen SANs beim Frameversand keine Duplikate und hängen somit auch kein RCT an die Frames. Erhält ein SAN ein Frame mit einem RCT wird dies als zusätzliches Padding ohne Bedeutung wahrgenommen. [25]

Eine zusätzliche Möglichkeit, um Geräte mit nur einem Netzwerkinterface an einem PRP-Netzwerk anzuschliessen, wäre über eine Redundancy Box (RedBox). Eine RedBox ist, wie ein DANP, über beide LANs im PRP-Netzwerk eingebunden und bietet weitere Anschlüsse für Geräte mit nur einem Interface. Ein solches Gerät, das über eine RedBox am PRP-Netzwerk teilnimmt, erscheint für die anderen Teilnehmer wie ein DAN und wird «Virtual Dual Attached Node» (VDAN) genannt. Somit fungiert die RedBox als Proxy für VDANs und hat aus Management-Gründen eine eigene IP-Adresse. [23, 25]

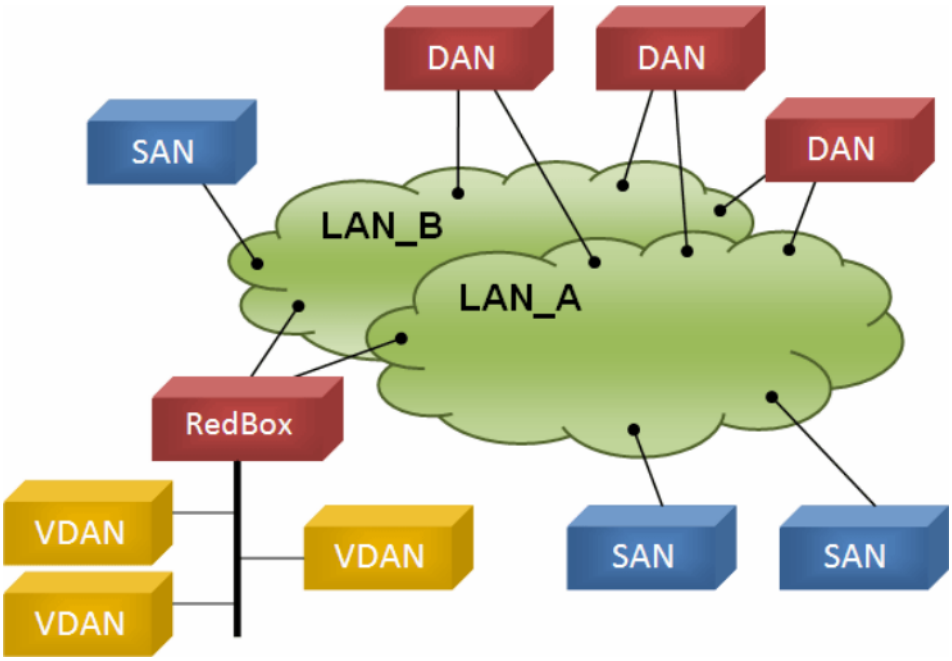


Abbildung 2.3: Beispielaufbau eines PRP-Netzwerks [23]

Durch diesen Ablauf und Aufbau wird garantiert, dass die Kommunikation zwischen den DANs und RedBoxen bei dem Ausfall eines LANs trotzdem bestehen bleibt, ohne Zeit für eine Umschaltung zu benötigen. Aufgrund dieser Eigenschaft wird PRP unter anderem als «seamless» oder «bumpless» bezeichnet. [24]

Da es sich bei PRP um ein Protokoll handelt, das von den MAC-Adressen der Netzwerkteilnehmer abhängig ist, wird kein IP-Routing unterstützt. Dies hat den Grund, weil ein IP-Router die Quell-MAC-Adresse im Ethernet-Frame ändert und ein Empfänger im PRP-Netzwerk die ursprüngliche Adresse benötigt, um Duplikate feststellen zu können. [43]

Für das Redundanzmanagement und das Überprüfen, ob andere DANs im Netzwerk anwesend sind, sendet jede LRE via Multicast regelmässig (laut Standard [54] alle 400 ms) sogenannte «PRP_Supervision»-Frames. Erhält eine LRE ein solches Frame, erzeugt es einen Eintrag in seiner Node-Tabelle mit der MAC-Adresse des betreffenden Knotens (dieser ist im «MacAddress»-Feld im «PRP_Supervision»-Frame aufgeführt). Die LRE einer RedBox sendet für jedes seiner VDANs ein solches Frame, allerdings werden die Informationen zur RedBox in zusätzlichen Feldern beigefügt. [54]

Erhält ein Knoten nach einer gewissen Zeit («NodeForgetTime», standardmässig 60’000 ms oder 1 min) von einem bestimmten Knoten kein «PRP_Supervision»-Frame mehr, jedoch noch andere Frames von dieser Quelle über lediglich ein LAN, wird der Status dieser Quelle im Knoten von «DAN» zu «SanA» oder «SanB» geändert (hängt vom verwendeten LAN ab). [54]

2.1.1 PRP-1 stack (Software-Implementation)

2.1.1.1 Übersicht

In dieser Arbeit wird PRP in Form einer Software-Implementation im Endgerät betrachtet. Bei der Software handelt es sich um einen vom ZHAW Institute of Embedded Systems entwickelten Open Source User-Mode-Stack für Linux. Dieser wird als Zwischenschicht realisiert, der über ein virtuelles Netzwerkinterface («PRP Driver») mit den höheren Schichten kommuniziert. Dabei erscheint der «PRP Driver» für diese Schichten wie ein normales Netzwerkinterface und stellt die LRE dar. [27]

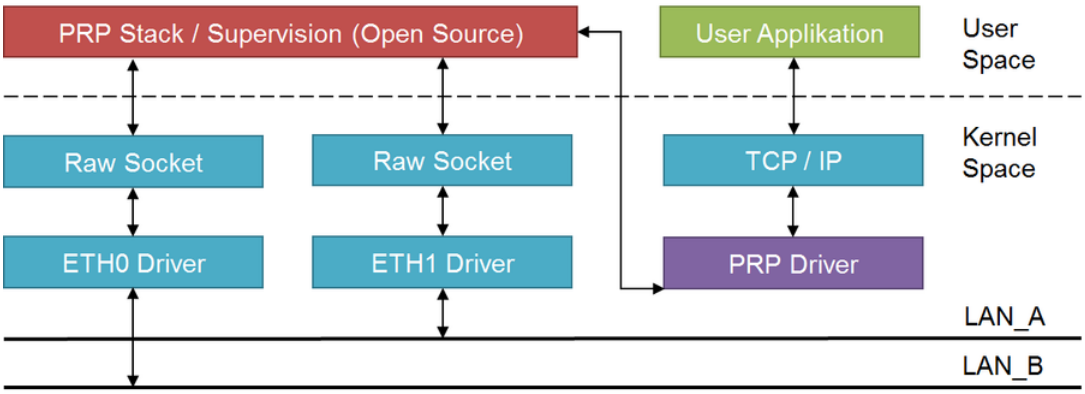


Abbildung 2.4: PRP-1 User Mode Stack [27]

2.1.1.2 Frame-Duplikate

Bei der Handhabung von Frame-Duplikaten sind zwei Aspekte von zentraler Beteutung, die genauer betrachtet werden müssen.

1. Ein neues Frame trifft ein. Der Software-Stack muss zu diesem Zeitpunkt kontrollieren, ob ein Duplikat dieses Frames bereits vorgängig auf dem anderen Interface eingetroffen ist. Falls dies zutrifft, muss das Frame verworfen werden, ansonsten muss das Frame, respektive dessen Meta-Daten in einer Datenstruktur über eine gewisse Zeit gehalten werden.

2. Da der Speicher einer Maschine endlich ist, müssen die Metadaten der erhaltenen Frames nach einer gewissen Zeitspanne wieder aus der Datenstruktur entfernt werden. Dies wird mittels Software-Timer erreicht. Nach Ablauf eines gewissen Timer-Intervalls muss die Datenstruktur, in der die Metadaten der Frames gehalten werden, aktualisiert und das älteste Frame verworfen werden.

2.1.1.3 Software-Timer

Die Software-Implementation des PRP-1 stack verfügt über 3 Software-Timer, welche beim Starten der Applikation initialisiert und gestartet werden. Ein Timer-Tick dauert 20 ms.

- **Aging-Timer** mit einem Intervall von **20 ms**. Mit jedem Timer-Tick der Applikation wird die Datenstruktur aktualisiert und die ältesten Frame-Meta-Daten werden gelöscht.

- **Statistics-Timer** mit einem Intervall von **1000 ms**. Nach Ablauf dieses Timer-Intervalls werden sämtliche Statistikwerte der gesamten PRP-Stack-Applikation ausgelesen und, falls so konfiguriert, auf der Konsole ausgegeben.

- **Supervision-Timer** mit einem Intervall von **2000 ms**. Ist dieses Timer-Intervall abgelaufen, wird auf beiden physischen PRP-Interfaces ein Supervision-Frame transmittiert.

2.1.1.4 Datenhaltung von Frame-Metadaten

Als Datenstruktur, um die Metadaten erstmals eingetroffener Frames zu halten, wird eine Hash-Table verwendet. Die Tabelle hat eine Grösse von 256 Einträgen. Aus der Sequenznummer eines Frames wird nun ein Hash-Wert gebildet. Dieser Hash-Wert repräsentiert sogleich die Index-Nummer des

Erzeugung des Hashwertes: Bei der Ankunft eines PRP-Frames wird die Sequenznummer, welche sich im Redundancy Control Trailer befindet, herausgelesen. Eine Sequenznummer umfasst 16 Bit.

2.1.1.5 Supervision-Frames

2.2 Offload-Mechanismen

Bei den Offload-Mechanismen handelt es sich um Features, die von gewissen Netzwerkkarten unterstützt werden und ein- oder ausgeschaltet werden können. In diesem Kapitel wird erläutert, was die unterschiedlichen Mechanismen im Groben bewirken. Hierbei liegen lediglich die Offload-Mechanismen, die von der in dieser Arbeit eingesetzen Hardware unterstützt werden, im Fokus.

RX/TX Checksum Offload

Wird der Checksum-Offload-Mechanismus aktiviert, werden die IP-, TCP- und UDP-Checksummen von der Netzwerkkarte anstelle dem Prozessor berechnet. Dieser Mechanismus lässt sich individuell für das Senden und Empfangen konfigurieren. [18]

Scatter-Gather

Ermöglicht es, dass man Daten von und in mehrere Buffer lesen und schreiben kann, die im Speicher voneinander getrennt sind (Die Speicherorte der Buffer sind in einem Vektor gespeichert). So wird die Anzahl von read- und write-System-Calls von Mehreren auf Einen gesetzt, was die CPU-Last erleichtert. [15]

TCP Segmenation Offload (TSO)

Bei TSO wird die Belastung der CPU minimiert, indem der Netzwerkkarte grosse Buffer angereiht werden, welche dann von dieser in separate Pakete gesplittet wird (Segmentierung). Ohne diese Segmentierung würde ein grosses TCP-Paket zuerst von der CPU segmentiert und einzeln an die NIC gesendet werden. Mit TSO ist es möglich, dass das grosse Frame direkt an die NIC gesendet wird und diese dann die Segmentierung vornimmt. [57]

UDP Fragmentation Offload (UFO)

Funktioniert ähnlich wie TSO mit dem Unterschied, dass hier das Fragmentieren von UDP-Datagrammen der NIC statt der CPU überlassen wird. [37]

Generic Segmentation Offload (GSO)

Beim Generic Segmentation Offload handelt es sich um eine Generalisierung von TSO (nicht auf TCP limitiert). Hierbei wird ein grosses Frame nicht in der NIC segmentiert, sondern vor der Übermittlungs-Routine im Treiber der Netzwerkkarte. [29, 36, 57]

Generic Recieve Offload (GRO)

GRO ist ein Mechanismus, der die eintreffenden Fragmente in ein grösseres Paket zusammenfasst, so dass vom Computer aus, der die Daten erhält, nicht mehr die einzelnen Fragmente, sondern ein grosses Frame sichtbar ist. [29]

Zusammengefasst kann man sagen, dass die Mechanismen TSO, UFO, GSO und GRO dazu dienen, mit einer MTU von 1500 Bytes umgehen zu können. Mit der ansteigenden Geschwindigkeit der Netzwerkkarten werden durch die Limitierung der MTU mehr Frames pro Sekunde erzeugt, was eine Steigerung der CPU-Belastung mit sich bringt. Damit diese Belastung nicht allzu stark ansteigt, werden durch die Offload-Mechanismen Operationen für die Frame-Behandlung auf die Netzwerkkarte ausgelagert. [29]

RX/TX VLAN Acceleration

Mittels VLAN Acceleration wird beim Senden / Empfangen der VLAN-Header in der NIC hinzugefügt / gekürzt. [31]

RX ntuple Filters and Actions

Dieses Feature ermöglicht es, dass die Netzwerkkarte Pakete je nach Filter in verschiedene Queues einordnet. So kann zum Beispiel einem Webserver eine eigene Recieve-Queue zugeordnet werden. Diese Filter und Queues können mit der Applikation ethtool konfiguriert werden. Es besteht die Möglichkeit unter anderem nach Protokoll, Quell- / Ziel-Adressen (MAC oder IP), Portnummern, VLANs, etc. zu filtern. [13, 49]

RX Hashing Offload

Anhand von «RX Hashing Offload» wird bei Erhalt eines Frames bereits in der Netzwerkkarte die Prüfsumme automatisch verifiziert. [34]

2.3 TCP-Window-Size

Beim Versand von TCP-Paketen kann neben dem Prozessor und der Netzwerkanbindung die TCP-Window-Size ein limitierender Faktor sein. Dabei handelt es sich um einen Wert, welcher von jedem Teilnehmer im TCP-Header mitgegeben wird. Dieser besagt in Bytes, wie viel an TCP-Paketen vom anderen gesendet werden kann, ohne eine Bestätigung zu erhalten. Wenn der Sender zum Beispiel vom Empfänger eine TCP-Window-Size von 16’384 Bytes erhält, versendet der Sender diese Anzahl an Bytes, hält dann den Sendevorgang an und wartet auf eine Quittierung bevor er mit dem Sendevorgang weiterfährt. [7]

Ein Beispiel zur Ermittlung der optimalen TCP-Window-Size x ist das Berechnen des Produkts von Bandbreite und RTT. Bei 100 MBit/s und einer RTT von 1ms (typischer RTT-Wert für ein maximal grosses Paket im lokalen Netzwerk via TCP [20]) ergibt dies folgende Rechnung:

$$x = \frac{100'000'000}{8}\, Byte/s \ * 0.001\, s = 12'500\, Byte$$

Somit kann der Sender hier 12’500 Bytes an Daten senden, bevor er auf eine Quittierung vom Empfänger wartet. Anhand diesem TCP-Window-Size-Wert müsste der Sender in diesem Beispiel nicht warten, bis er die Quittierung für die ersten 6’250 Bytes (Bandbreite multipliziert mit der Dauer eines Weges: 0.5ms) vom Empfänger erhält, sondern könnte die Zeit, in welcher die Quittierung übertragen wird, für das Senden weiterer 6’250 Bytes nutzen. Dies liegt daran, dass die RTT als Faktor miteinbezogen wurde, welche den Hin- und Rückweg eines Pakets beinhaltet. [7]

Ob die TCP-Window-Size ein limitierender Faktor beim TCP-Verkehr dieser Arbeit darstellt wird in Kapitel 3.3.4.2 auf Seite 96 untersucht.

2.4 /proc-Dateisystem

Bei dem /proc-Dateisystem handelt es sich um ein Pseudo-Dateisystem, welches als Schnittstelle für Datenstrukturen des Linux-Kernels dient. Üblicherweise ist dieses Dateisystem in einem Linux-Betriebssystem unter dem Pfad /proc eingebunden. [48]

Jedes mal, wenn auf eine Datei aus dem /proc-Dateisystem ein read()-Systemcall ausgeführt wird, werden die aktuellsten Statistiken ausgelesen. Der Kernel besitzt einen Memory-Bereich, wo sämtliche Statistiken abgelegt werden. Zu diesem Memory-Bereich gehören auch Adressräume, die von Gerätetreibern von so genannten «Memory Mapped Hardware Devices» genutzt werden, um direkt auf Hardware-Ressourcen (Register) zugreifen zu können, dies wird mittels Memory-Mapped-I/O-Verfahren realisiert. /proc ist im Wesentlichen und sehr vereinfacht gesagt ein Zeiger auf diesen Kernel-Memory-Bereich.

Die Datenstrukturen können ausgelesen werden, um so die aktuellsten System- und Prozess-Informationen vom laufenden Linux-Kernel erhalten zu können. Zudem kann die Konfiguration des Linux-Kernels, während dieser in Betrieb ist, über das /proc-Dateisystem geändert werden. [41]

Jeder Prozess erhält im /proc-Dateisystem einen Ordner, der die Prozess-ID-Nummer als Namen trägt. In einem solchen Ordner (z.B. /proc/123, bei einem Prozess mit der ID 123) ist eine umfangreiche Auswahl an Daten zum entsprechenden Prozess aufzufinden. Nicht-prozessbezogene Informationen findet man im Wurzelverzeichnis des /proc-Dateisystems. Zum Beispiel befinden sich detaillierte Informationen zum Arbeitsspeicher unter /proc/meminfo. [41]

Teil II

Engineering

3 Vorgehen / Methoden

3.1 Aufbau der Testumgebung

3.1.1 Hardware

3.1.1.1 Server

——————————— Bild Server ———————————

Die Testumgebung besteht aus 3 Servern mit je 3 Netzwerkanschlüssen. Davon werden je 2 für die PRP-Umgebung verwendet, wobei der dritte Anschluss lediglich für administrative Zwecke verwendet wird. So ist es möglich, von Aussen auf die Server zugreifen zu können, ohne das der Datenverkehr, der über die PRP-Schnittstellen transportiert wird, durch administrative Netzwerkprotokolle beeinflusst wird. Dies soll einer Verfälschung der Messergebnisse vorbeugen. Sämtliche Offload-Mechanismen (siehe Kapitel 2.2 auf Seite 19) sind deaktiviert, wobei dessen Einflüsse später untersucht werden (siehe Kapitel 3.3.4.10 auf Seite 103).

Alle 3 Server sind mit identischer Hardware ausgestattet. Die Server verfügen über die nachfolgend gelisteten technischen Merkmale:

Eigenschaft	
Hersteller / Name	HP ProLiant DL140
CPU	Intel(R) Xeon(TM) CPU 2.40GHz
L1- / L2- / L3-Cache	8KiB / 512KiB / –
Wortbreite	32 Bit
Arbeitsspeicher (gesamt)	2 GiB
Architektur	i686 / x86
Festplatte	80 GiB (Software-RAID-1)
Netzwerkanschluss «eth0»	1 GBit/s Broadcom Corporation NetXtreme BCM5704 Gigabit Ethernet (rev 02)
Netzwerkanschluss «eth1»	1 GBit/s Broadcom Corporation NetXtreme BCM5704 Gigabit Ethernet (rev 02)
Netzwerkanschluss «eth2»	100 MBit/s 3Com Corporation 3c905C-TX/TX-M [Tornado] (rev 74)
Betriebssystem	Debian 7.8.0 32bit (i686 / x86 CPU) Version mit Non-Free-Firmware inklusive (Broadcom Tigon3 Treiber)

Tabelle 3.1: Hardware-Eigenschaften der Server

3.1.1.2 Switches

——————————— Bild Switch ———————————

Um die Server miteinander zu verbinden, werden 3 8-Port-HP-Switches verwendet, die über eine Kapazität von 1 GBit/s verfügen. 2 der Switches bilden die physischen Netzwerke A und B. Diese werden dafür verwendet, um das virtuelle PRP-Netzwerk zu bilden. Der 3. Switch dient dazu, ein autonomes Netzwerk zu bilden, das für administrative Zwecke verwendet werden kann, und es ermöglicht, via Gateway eine Verbindung zu einem anderen LAN herzustellen.

3.1.2 Netzwerke

Um ein PRP-Netzwerk mit den 3 Servern aufzubauen, sind 2 physische Netzwerke nötig, die dann ein virtuelles PRP-Netzwerk bilden. Im Folgenden wird grafisch aufgezeigt, wie die Testumgebung aufgebaut ist.

3.1.2.1 Physische Netzwerke

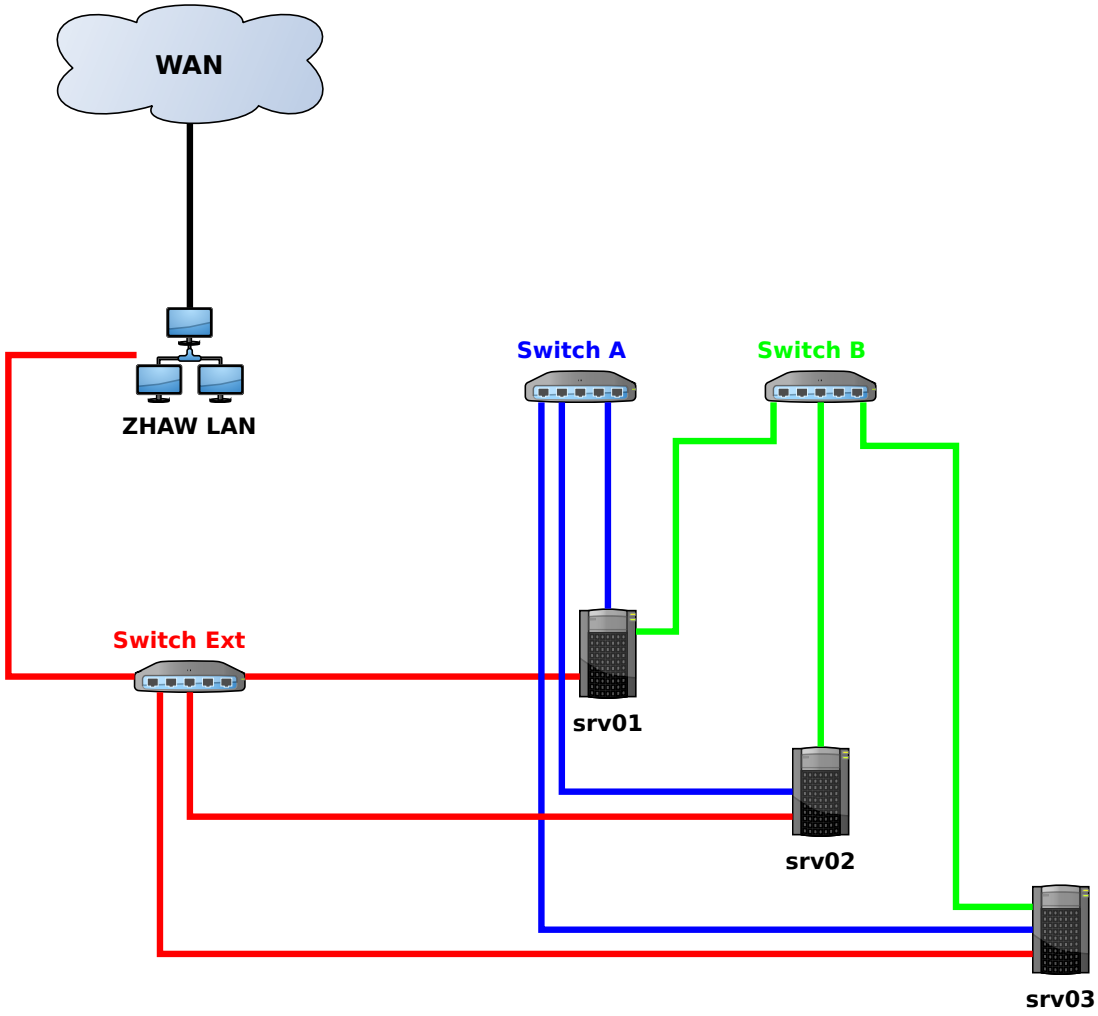


Abbildung 3.1: Aufbau der Testumgebung - Physisches Netzwerk

	Netzwerk A	Netzwerk B	Netzwerk Ext
Netzwerk	192.168.1.0	192.168.2.0	192.168.99.0
srv01	192.168.1.1	192.168.2.1	192.168.99.1
srv02	192.168.1.2	192.168.2.2	192.168.99.2
srv03	192.168.1.3	192.168.2.3	192.168.99.3

Tabelle 3.2: Konfigurationsdaten - Physisches Netzwerk

IPv4-Konfiguration: Für das Administrationsnetzwerk (192.168.99.0) wird der Gateway mit der IP-Adresse 192.168.99.100 verwendet. Damit wird ein Zugang von einem anderen LAN aus auf die Server ermöglicht.

3.1.2.2 PRP-Netzwerk

Ein PRP-Netzwerk basiert grundsätzlich auf dem Konzept, dass aus zwei physischen Netzwerken ein virtuelles Netzwerk gebildet wird, um die gewünschte Redundanz zu erreichen. Beispielsweise wird aus zwei physischen Netzwerk-Interfaces ein Virtuelles. Die folgende Abbildung zeigt, wie die physischen Verbindungen zweier Netze zu einer virtuellen Verbindung eines PRP-Netzwerks werden.

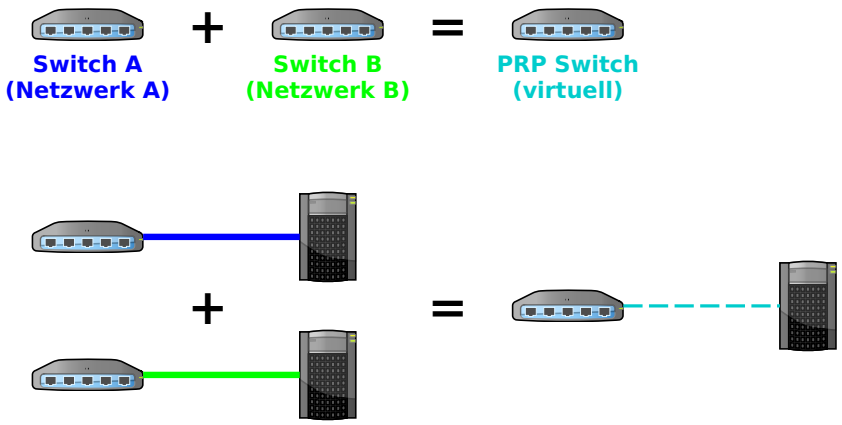


Abbildung 3.2: Zusammenhang physische und virtuelle Verbindung

Daraus lässt sich der Aufbau des virtuellen PRP-Netzwerks ableiten und folgendermassen aufzeigen:

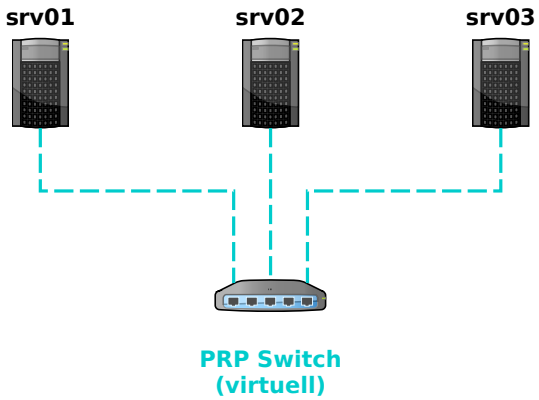


Abbildung 3.3: Aufbau der Testumgebung - PRP-Netzwerk

IPv4-Konfiguration:

	PRP
Netzwerk	192.168.0.0
srv01	192.168.0.1
srv02	192.168.0.2
srv03	192.168.0.3

Tabelle 3.3: Konfigurationsdaten - PRP-Netzwerk

3.1.2.3 Netzwerkgeschwindigkeiten

Die in die Server eingebauten Broadcom-Interfaces haben einen maximalen Datendurchsatz von 1 GBit/s (OSI-Layer 1). Im Rahmen dieser Arbeit wird die Netzwerkgeschwindigkeit auf 100 MBit/s reduziert. Dies wird erreicht durch den Einsatz der Software ethtool. Die Reduzierung lässt sich dadurch erklären, dass für die Messungen Hardware im Einsatz ist, die über Netzwerk-Interfaces verfügt, welche lediglich Geschwindigkeiten von maximal 100 MBit/s verarbeiten können. Ferner sind auf diese Weise Performanz-Engpässe einfacher zu erörtern (Netzwerk-Interface oder CPU). Netzwerkkarten limitierte Engpässe entstehen schneller mit 100 MBit/s als mit 1 GBit/s.

Das Festlegen der Geschwindigkeiten des Interfaces «eth0» geschieht mittels folgendem ethtool-Befehl:

```
ethtool -s eth0 speed 100 duplex full autoneg on
```

Die maximal mögliche Geschwindigkeit des Interfaces wird hier auf 100 MBit/s reduziert. Desweiteren wird die Autonegotiation eingeschaltet. Dies ist notwendig, da im Falle ausgeschalteter Autonegotiation beispielsweise eine ECI-Box (siehe 3.20 auf Seite 39) keine Verbindung mehr herstellen kann mit den Broadcom-Interfaces der Server.

3.1.2.4 /proc-Netzwerkstatistiken

Um möglichst genaue Messresultate zu erhalten, sollten die Statistiken der Netzwerk-Interfaces möglichst zeitnah im /proc-Dateisystem aktualisiert werden. Wie sich das Aktualisierungsverhalten von Neztwerk-Interfaces charakterisiert, hängt von den Netzwerk-Interfaces (Hardware) selbst ab.

Bei den in dieser Arbeit druchgeführten Messungen werden die Interface-Statistiken von «eth0» und «eth1» (siehe 3.1 auf Seite 24) betrachtet. Für Messungen bezüglich des PRP-Protokolls wird durch den PRP-1 stack ein virtuelles Netzwerk-Interface erstellt «prp1». Das virtuelle Interface des PRP-1 stacks wiederum benutzt die physische Netzwerk-Hardware «eth0» und «eth1», um Daten zu empfangen, respektive zu transmittieren.

Bei den physischen Interfaces «eth0» und «eth1» handelt es sich um Hardware des Herstellers Broadcom. Die Netzwerkkarten bieten die Möglichkeit, das Aktualisierungs-Intervall der Statistiken manuell zu modifizieren. Im Auslieferungszustand haben die Broadcom-Interfaces ein Aktualisierungsintervall von einer Sekunde eingestellt.

Die Modifikation des Statistik-Intervalls lässt sich mit ethtool wie folgt durchführen:

```
ethtool -C eth0 stats-block-usecs 100
```

Die vorgängig gezeigte Kommandozeile setzt ein Statistik-Aktualisierungs-Intervall von $100\mu s$ für das Interface «eth0». Damit die Änderung aktiv wird, muss anschliessend das Interface «eth0» neu gestartet werden. Um sicherzustellen, dass die Änderungen übernommen wurden, können die Einstellungen des entsprechenden Netzwerk-Interfaces abgefragt werden:

```
ethtool -c eth0
```

Was eine Ausgabe folgender Form erzeugt:

```
Coalesce parameters for eth0:
Adaptive RX: off  TX: off
stats-block-usecs: 100
sample-interval: 0
pkt-rate-low: 0
pkt-rate-high: 0

rx-usecs: 20
...
```

Es wurde ein Aktualisierungs-Intervall mit einer Dauer von $100\mu s$ gewählt. Mit kürzeren Intervallen wäre der maximale Messfehler noch kleiner, jedoch besteht die Gefahr, dass die Performanz der Netzwerk-Interfaces unter dieser Konfiguration beeinträchtigt würde, was Geschwindigkeitseinbussen in der Datenübermittlung zur Folge hätte.

Das Kommando:

```
ethtool -i eth0
```

Liefert Informationen über den Gerätetreiber des Interfaces mit dem Namen «eth0»:

```
driver: tg3
version: 3.121
firmware-version: 5704-v3.26
bus-info: 0000:02:00.0
supports-statistics: yes
supports-test: yes
supports-eeprom-access: yes
supports-register-dump: yes
supports-priv-flags: no
```

Über die Zeile, welche die Bus-Info ausgibt, kann in der Datei /proc/iomem erörtert werden, auf welchen Bereich im Hauptadressraum des Betriebssystems die Speicheradressen des Interfaces «eth0» übersetzt werden. Über diesen Mechanismus wird vom Betriebssystem beispielsweise auf die statistischen Daten des Interfaces zugegriffen.

```
cat /proc/iomem | grep 0000:02:00.0
```

Adressbereiche für den Bus 0000:02:00.0:

```
febb0000-febbffff : 0000:02:00.0
febc0000-febcffff : 0000:02:00.0
```

$$febcffff_h - febb0000_h + 00000001_h = 20000_h = 131072_d = 128kByte$$

Ein Broadcom Interface verfügt über einen Speicher von 128 kByte.

3.1.3 Standorte

Die Testumgebung wurde in den Räumlichkeiten des Gebäudes TE der ZHAW in Winterthur aufgebaut. Die Server wurden in einem nicht öffentlichen Raum (TE524) platziert. Die beiden Switches (A und B) befinden sich im Arbeitsraum (TE523). Dies ermöglicht es, Messgeräte sowie simulierte Verzögerungen einzubauen. Ebenfalls können so weitere Geräte an die Netzwerke A oder B angeschlossen werden.

Das nachfolgende Schema zeigt die detaillierte, physische Verkabelung der gesamten Testumgebung:

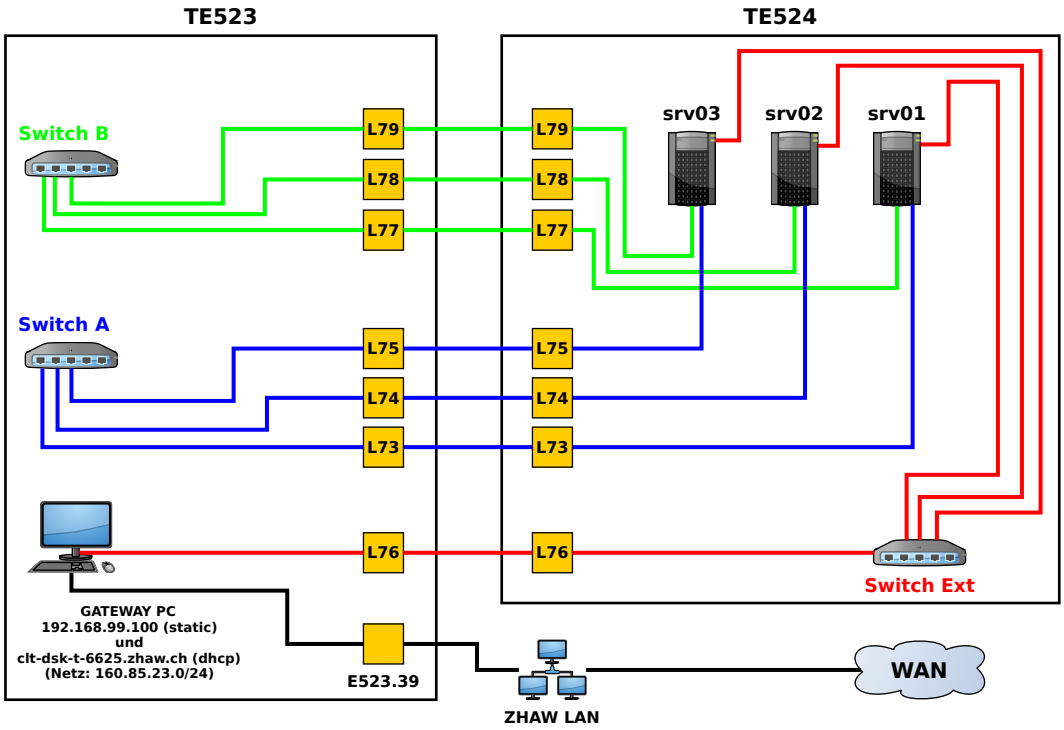


Abbildung 3.4: Aufbau der Testumgebung - Detaillierte Verkabelung

3.2 Evaluierung der Tools

Für die Performance-Messungen werden auch bereits bestehende Tools verwendet. Da bereits eine Vielzahl solcher Tools existiert, wird in diesem Abschnitt versucht, die Auswahl zu verwendender Tools einzugrenzen. Für diese Selektierung werden die Aspekte berücksichtigt, welchen gemäss der Aufgabenstellung [22] besondere Beachtung beigemessen werden soll.

Im spezifischen Fall mit der PRP-Testumgebung sind dies:

- Effekt von Laufzeitunterschieden zwischen LAN_A und LAN_B
- Zeitweiser Ausfall eines Netzwerkpfades
- Abhängigkeit vom verwendeten Protokoll
- Einfluss nicht entfernter Duplikate auf Funktion und Performance
- Auswirkung auf Applikationen, wenn Frames out-of-sequence ankommen

Damit diese Aspekte untersucht werden können, muss eine Auswahl an diversen Applikationen oder Geräten zur Verfügung stehen, um die Messungen durchführen zu können. Diese Hilfsmittel müssen in der Lage sein, die folgenden Tätigkeiten durchzuführen:

- Generierung von Netzwerktraffic über diverse Protokolle (u.a. TCP und UDP)
- Messen von Ressourcennutzung und Performanceparametern
- Simulieren von Fehlern in der Datenübertragung

Es folgt eine Auflistung von Hilfsmitteln inklusive deren Beschrieb, Funktionen und Grund, warum es in dieser Arbeit verwendet oder nicht verwendet wird. Dabei wurden zuvor Hilfsmittel ausgeschlossen, die entweder eine grafische Oberfläche benötigen, kaum bekannt oder dessen aktuellste Version älter als 2012 sind. Im nächsten Kapitel wird die engere Auswahl der gefundenen Tools aufgelistet und evaluiert.

3.2.1 Generierung von Netzwerktraffic

Name	**flowgrind [8]**
Beschrieb	Erzeugt Netzwerkverkehr über TCP für das Testen und Messen von TCP/IP-Stacks. Es wird damit geworben, dass mit flowgrind Transport-Layer-Informationen ausgeben kann, die üblicherweise interne TCP/IP-Stack-Informationen sind. flowgrind wird auf mehreren Rechnern ausgeführt. Auf den Computern, zwischen denen die Messung stattfinden soll, wird der flowgrind-Deamon gestartet, während auf einem weiteren Computer den flowgrind-Controller betreibt, der die Messdaten sammelt.
Funktionen	flowgrind ermöglicht es, mehrere Flüsse mit gleichen oder unterschiedlichen Einstellungen gleichzeitig zu betreiben. Für das Testen und den Controller können verschiedene Netzwerkinterfaces zugeordnet werden.
Wird verwendet	Nein
Grund	Die Aufgabenstellung [22] verlangt neben dem Messen von TCP- u.a. auch das Messen von UDP-Verbindungen. Es wird bevorzugt, dass diese Verbindungen mit dem Selben Tool betrachtet werden, um einen klaren Vergleich zwischen den Protokollen zu ermöglichen.

Tabelle 3.4: Hilfsmittel zur Generierung von Netzwerktraffic: flowgrind

Name	**iperf3 [14]**
Beschrieb	Dient zur Messung der Netzwerkbandbreite und ermöglicht Trafficgenerierung über TCP, UDP oder SCTP. iperf3 ist eine von Grund auf neu implementierte Lösung von iperf. Das Tool wird auf mehreren Computern gleichzeitig ausgeführt, mal als Server und mal als Client.
Funktionen	Unter anderem bietet iperf3 neben den im Beschrieb erwähnten Funktionen einen Zero-Copy-Modus an und ermöglicht das Versenden von vordefinierten Streams.
Wird verwendet	Ja
Grund	Die neuste Version von iperf3 wurde am 09.01.2015 veröffentlicht, was darauf schliessen lässt, dass das Tool derzeit gepflegt wird. Des Weiteren wurde den Autoren dieser Arbeit bei den Besprechungen empfohlen, diese Applikation zu verwenden.

Tabelle 3.5: Hilfsmittel zur Generierung von Netzwerktraffic: iperf3

Name	**netperf [46]**
Beschrieb	netperf ist ähnlich wie iperf3 und bietet auch Generierung von Netzwerktraffic und Messung von Verbindungen über TCP, UDP und SCTP an.
Funktionen	Weitere Features von netperf sind Histogram-Support und Informationen über die Nutzung der CPU.
Wird verwendet	Nein
Grund	Die neuste Version von netperf wurde am 19.06.2012 veröffentlicht. Da netperf ähnlich zu iperf3, jedoch weniger aktuell ist, wird iperf3 vorgezogen.

Tabelle 3.6: Hilfsmittel zur Generierung von Netzwerktraffic: netperf

3.2.2 Messen von Ressourcennutzung und Performanceparametern

Name	**atop [19]**
Beschrieb	Performance-Monitor, der über die Aktivität aller Prozesse bezüglich CPU, RAM und Speicher berichten kann.
Funktionen	Mit atop wird der Ressourcenverbrauch aller Prozesse (inklusive denen, die während der Aufzeichnung beendet werden) aufgezeichnet. Des Weiteren können Berichte generiert werden, um die Daten später einzusehen.
Wird verwendet	Nein
Grund	Erstellt regelmässig Berichte, welche in dieser Arbeit nicht von Nöten sind.

Tabelle 3.7: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: atop

Name	**bwm-ng [55]**
Beschrieb	Zeichnet die Netzwerk- und Disk-I/O-Bandbreite auf.
Funktionen	Bietet unterschiedliche Darstellungsfunktionen, u.a. auch Exportierungen nach CSV & HTML und ermöglicht es, in Intervallen zu messen.
Wird verwendet	Ja
Grund	Mit bwm-ng kann die Netzwerkbelastung in Intervallen gemessen sowie auch dessen Output als CSV-Datei gespeichert werden.

Tabelle 3.8: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: bwm-ng

Name	**NetHogs [5]**
Beschrieb	Zeigt den Netzwerkverkehr pro Prozess an.
Funktionen	Die Download- und Upload-Geschwindigkeit von TCP-Verbindungen wird pro Prozess angezeigt.
Wird verwendet	Nein
Grund	Auf der NetHogs-Webseite [5] ist aufgeführt, dass die Applikation noch nicht korrekt auf Computern funktioniert, die über mehrere IP-Adressen verfügen. Schon aus diesem Grund eignet sich NetHogs nicht für den Einsatz in dieser Arbeit.

Tabelle 3.9: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: NetHogs

Name	**ntop [35]**
Beschrieb	Weist das Geschehen im Netzwerk auf und zeigt Informationen bezüglich dem von jedem Knoten generierten und erhaltenen Netzwerkverkehr auf.
Funktionen	ntop kann IP- und Nicht-IP-Traffic kombinieren und bietet seine Informationen via integriertem Webserver an, weshalb ein Webbrowser benötigt wird, um an die Informationen heran zu kommen.
Wird verwendet	Nein
Grund	Dadurch, dass ntop seine Informationen lediglich über den eigenen Webserver anbietet, der Einiges an zusätzlichen Ressourcen auf den DANs beanspruchen würde, wird auf den Einsatz von ntop verzichtet.

Tabelle 3.10: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: ntop

Name	**pmap [2]**
Beschrieb	Die Memory Map eines Prozesses kann mittels pmap angezeigt werden. So lässt sich genau feststellen was und wie viel auf dem Arbeitsspeicher durch den Prozess beansprucht wird.
Funktionen	Jeder vom Prozess beanspruchten Bereich wird u.a. mit dessen Startadresse, Grösse und Berechtigungen aufgewiesen.
Wird verwendet	Ja
Grund	Mittels pmap können ausführliche Aussagen über die Arbeitsspeichernutzung eines Prozesses gemacht werden.

Tabelle 3.11: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: pmap

Name	**strace [30]**
Beschrieb	Diese Applikation ermöglicht es zu sehen, welche System-Calls von einem bestimmten Prozess aufgerufen werden.
Funktionen	Zeigt alle Parameter, die bei einem System-Call übergeben wurden, und die Rückgabewerte dieser System-Calls an.
Wird verwendet	Ja
Grund	Anhand von strace kann das Verhalten zwischen dem PRP-1 User Mode Stack (siehe Kapitel 2.1.1 auf Seite 17) und dem Betriebssystem untersucht werden.

Tabelle 3.12: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: strace

Name	**sysstat [47]**
Beschrieb	Sammlung an Tools, um die Systemleistung und -Aktivitäten zu überwachen, welche folgende Applikationen beinhaltet: • iostat: Legt Statistiken zur CPU und Input / Output von Geräten, Partitionen und Netzwerk-Dateisystemen dar. • mpstat: Weist individuelle oder kombinierte, prozessorbezogene Werte auf. • pidstat: Bringt Informationen zu einzelnen Prozessen bezüglich u.a. Input / Output, CPU, Arbeitsspeicher hervor. • sar: Sammelt, veranschaulicht und speichert System-Aktivitäts-Informationen zu CPU, Arbeitsspeicher, Speicher, Interrupts, Netzwerkinterfaces und weiteren Komponenten. • sadc: Backend von sar, das für das effektive Sammeln der Informationen zuständig ist. • sa1: Trägt Werte zur System-Aktivität zusammen und speichert sie in eine Datei ab, die täglich ausgewechselt wird. sa1 ist ein Frontend zu sadc und wurde mit dem Gedanken entwickelt, regelmässig ausgeführt zu werden. • sa2: Schreibt eine Zusammenfassung der System-Aktivitäten, die einen Tag umfasst. Wie sa1 wurde sa2 dafür entwickelt, dass es regelmässig ausgeführt wird. • sadf: Bietet die von sar gesammelten Daten in unterschiedlichen Formaten (u.a. CSY und XML) an, um sie in Datenbanken zu laden oder mittels einem Tabellenkalkulationsprogramm zu illustrieren. • nfsiostat: Zeigt Statistiken bezüglich Input / Output bei Netzwerk-Dateisystemen auf. • cifsiostat: Belegt Informationen zu CIFS.
Funktionen	Die Tools in sysstat dienen dazu, um durch eine regelmässige Ausführung eine detaillierte Ansammlung an Statistiken zur System-Aktivität für jeden Tag anlegen zu können. Dennoch ist es möglich, Bestandteile der Toolsammlung einzeln auszuführen, ohne eine grosse Statistiksammlung anlegen zu müssen.
Wird verwendet	Ja
Grund	Zwar wird die sysstat-Sammlung nicht komplett verwendet, jedoch erweisen sich einzelne Komponenten wie z.B. pidstat als nützlich. Regelmässige Statistiken werden somit nicht angelegt, was die Verwendung der Tools aber nicht ausschliesst.

Name	**tcpdump [53]**
Beschrieb	Zeichnet den Netzwerkverkehr auf und stellt dessen Inhalt dar.
Funktionen	Es lassen sich Filter deklarieren, mit denen man festlegen kann, welcher Teil vom Netzwerkverkehr von Interesse ist. Unter anderem können die Werte in eine Datei ausgelagert oder die Ausgabe modifiziert werden.
Wird verwendet	Nein
Grund	Da tshark alle Funktionalitäten von tcpdump abdeckt und dazu noch mehr bietet, wird tshark gegenüber tcpdump vorgezogen.

Tabelle 3.14: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: tcpdump

Name	**top [3]**
Beschrieb	Performance-Monitor, der über die Aktivität aller Prozesse bezüglich CPU und RAM berichten kann.
Funktionen	Der Ressourcenverbrauch kann in Echtzeit angezeigt werden. Dazu wird ein «Batch mode» angeboten, mit dem es möglich ist, die Werte zu anderen Applikationen oder Dateien zu leiten.
Wird verwendet	Ja
Grund	top ist weit verbreitet und bietet die Informationen bezüglich Ressourcenverbrauch, die für diese Arbeit benötigt werden.

Tabelle 3.15: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: top

Name	**tshark [17]**
Beschrieb	Kann wie tcpdump den Inhalt des Netzwerkverkehrs aufzeichnen und analysieren.
Funktionen	Bietet ähnliche Funktionen wie tcpdump und noch mehr, da tshark in der Lage ist, die verwendeten Protokolle im Netzwerktraffic aufzuzeigen und zu analysieren.
Wird verwendet	Ja
Grund	Mit tshark sind gegenüber tcpdump einiges mehr an Filtermöglichkeiten gegeben. Des Weiteren wird PRP offiziell von den tshark-/Wireshark-Entwicklern unterstützt. [42]

Tabelle 3.16: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: tshark

Name	**valgrind [33]**
Beschrieb	valgrind dient dazu, Linux-Applikationen zu debuggen und u.a. Memory-Leaks zu entdecken.
Funktionen	Stellt automatisch Fehler bezüglich Memory-Management und Threading fest. Dazu macht valgrind ein detailliertes Profiling einer Applikation, um bei der Suche nach Bottlenecks zu helfen.
Wird verwendet	Ja
Grund	Wird verwendet, um in eigens entwickelten C-Applikationen nach Fehlern zu suchen.

Tabelle 3.17: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: valgrind

3.2.3 Weitere Tools

Name	**arping [39]**
Beschrieb	Arbeitet ähnlich wie ping, jedoch auf dem Ethernet-Layer.
Funktionen	Mittels arping können ARP-Requests versendet werden. Dabei wird angezeigt, wie lange für eine Antwort benötigt wurde.
Wird verwendet	Ja
Grund	Anhand von arping kann das Netzwerk auf Layer 2 untersucht werden.

Tabelle 3.18: Weitere Hilfsmittel: arping

Name	**Calnex Paragon [10]**
Beschrieb	Störungs-Generator, den man bei einem Netzwerkanschluss dazwischen einschleusen kann.
Funktionen	Es können Verzögerungen, Alarme und Paket-Korruptionen generiert und der Paketfluss überwacht werden.
Wird verwendet	Ja
Grund	Anhand diesem Generator ist es möglich, Laufzeitunterschiede zwischen Netzwerk A und B in der PRP-Umgebung zu erzeugen.

Tabelle 3.19: Weitere Hilfsmittel: Calnex Paragon

Name	**Ethernet Cable Interceptor (ECI) [6, 26]**
Beschrieb	Netzwerkperipherie, welche bei einem 10/100 Base-TX Netzwerkanschluss dazwischen geschaltet werden kann und benötigt somit weder Modifikationen an Hardware oder zusätzliche Treiber.
Funktionen	Mit dem ECI können Framefehler und -Verluste erzeugt werden. Man kann gezielt reproduzierbare Störungen in einem Netzwerk generieren.
Wird verwendet	Ja
Grund	Durch das Erzwingen von Frameverlust kann die Stabilität von PRP und dessen Einfluss auf die Performance überprüft werden.

Tabelle 3.20: Weitere Hilfsmittel: Ethernet Cable Interceptor (ECI)

Name	**vmstat [21]**
Beschrieb	Zeigt Informationen zu u.a. Prozessen, Arbeitsspeicher, Traps, etc.
Funktionen	Stellt Durchschnitte seit dem letzten Neustart oder innerhalb einer bestimmten Zeitspanne dar.
Wird verwendet	Nein
Grund	Mittels vmstat können keine Prozesse einzeln analysiert werden.

Tabelle 3.21: Weitere Hilfsmittel: vmstat

3.2.4 Eigene Applikationen

3.2.4.1 Messung von CPU- und Netzwerk-Last «meas»

Es wird eine eigene Anwendung in C (mit dem Namen «meas», kurz für «measurement») entwickelt, bei der die CPU-Last einzelner Prozesse evaluiert wird. Des Weiteren ermittelt meas neben der CPU-Last, die System- und Usertime eines Prozesses. Meas unterstützt nur das Überwachen von Single-Threaded-Prozessen. Diese Limitierung wurde aufgrund der Gegebenheit, dass im Rahmen dieser Arbeit nur Single-Threaded-Prozesse beobachtet werden, in Kauf genommen.

Zusätzlich unterstützt meas zwei Modi im Hinblick auf das Testszenario 02 (siehe 4.2 auf Seite 113):

- Ermitteln von Statistiken aller PRP-involvierten Netzwerk-Interfaces
  [z. B.: «prp1» (virtuell), «eth0» (physisch), «eth1» (physisch)]
- Ermitteln von Statistiken eines einzeln physischen Netzwerk-Interfaces (ohne PRP)
  [z. B.: «eth0» (physisch)]

Netzwerkstatistiken, Quelldaten Die Messresultate bezüglich erhaltener und transmittierter Bytes sowie die Berechnungnen von Bitraten basieren ausschliesslich auf Werten, welche das Betriebssystem vom Data-Link-Layer aufzeichnet.

Übersicht

Die nachfolgende Grafik, soll eine Übersicht um den Gesamtumfang der Software meas illustrieren:

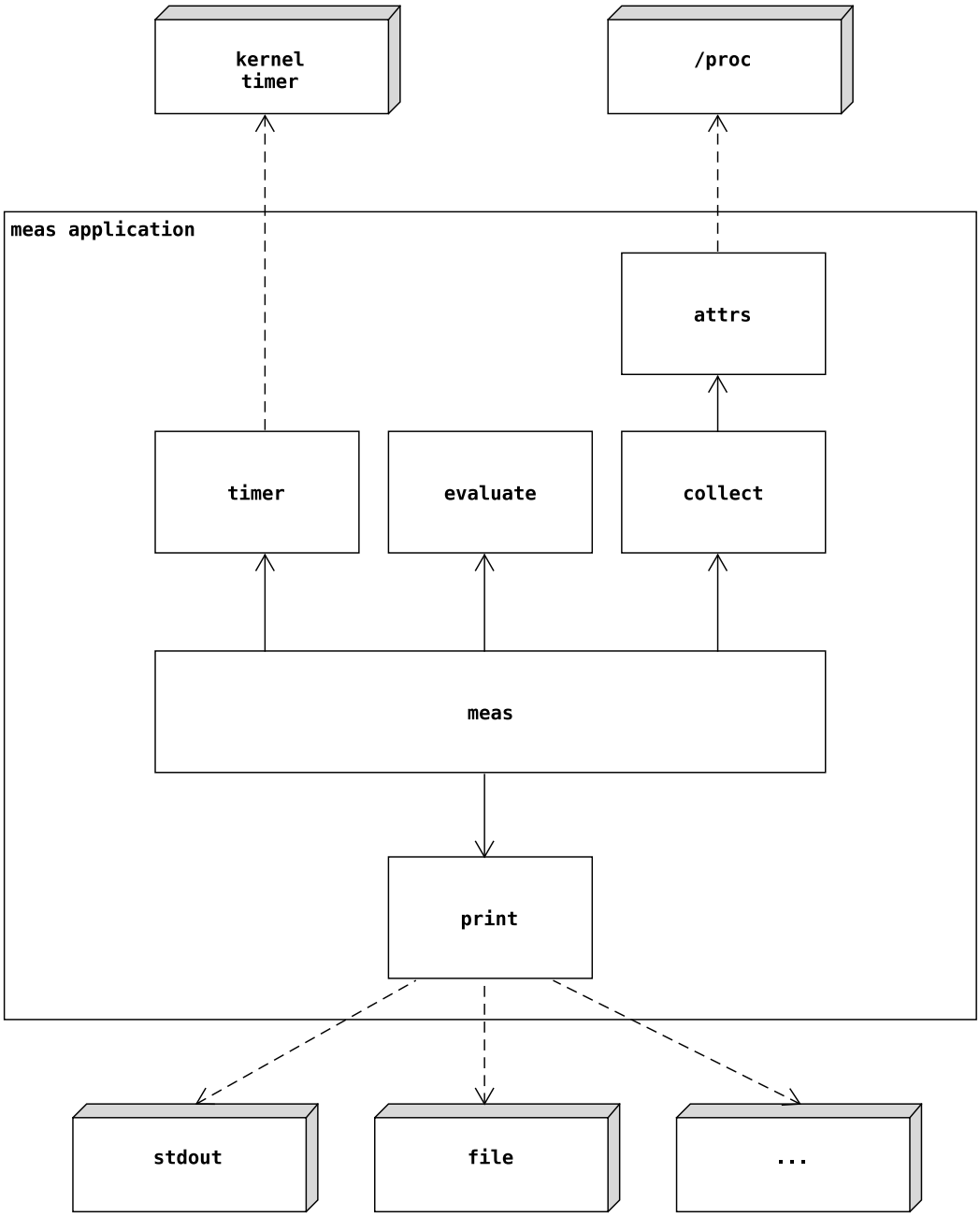


Abbildung 3.5: Gesamtübersicht meas-Applikation

Der Betriebssystem-Kern legt periodisch statistische Daten in ein Dateisystem ab, welches beim Bootprozess des Betriebssytems im Arbeitsspeicher angelegt wird, das /proc-Dateisystem, siehe 2.4 auf Seite 21. Aus diesem Dateisystem sollen alle relevanten statistischen Werte für die meas-Applikation gelesen und ausgewertet werden.

Beim Starten von meas gibt der Benutzer eine Intervall-Dauer sowie eine Anzahl von Intervallen und die zu überwachende Prozess-ID an. Im Attribut-Modul (attrs), wird spezifiziert, welche Zeile und Spalte aus welcher Datei im /proc-Dateisystem gelesen werden soll.

Das Sammel-Modul (collect), ist dafür zuständig, die im Attribut-Modul spezifizierten Werte in ensprechende Datenstrukturen abzulegen, sowie die Werte in Datentypen umzuwandeln, mit denen arithmetisch gerechnet werden kann. Das Timer-Modul (timer) wird verwendet, um die effektive Intervall-Dauer und Messfehler, damit ist die Zeit gemeint, welche aufgewendet werden muss, um von den entsprechenden Dateien im /proc-Dateisystem zu lesen. Es wird ein Timer verwendet, der eine Auflösung von 1 Mikrosekunde bereitstellt. Das Auswertungs-Modul (evaluate) hat die Aufgabe, gesammelte Statistiken in Form von Intervall-Resultaten auszuwerten und zusätzlich eine Gesamtstatistik zu berechnen. Im Print-Modul (print), werden ausgewertete Intervall-Daten, wie auch die Gesamtstatistik in eine für den Benutzer lesbare Form gebracht und können an eine beliebige Ausgabe geleitet werden, beispielsweise Textdateien, Konsole, Drucker, etc.

meas-Datentypen
Um zusammengehörende Daten sinnvoll zu gliedern werden C-Structs verwendet:

C-Datentyp	Kurzbeschrieb
cmd_args_t	Enthält alle Argumente, die der Benutzer beim Starten der meas-Applikation via Kommandozeile übergeben hat (Prozess-ID, Intervallzeit, Anzahl Intervalle, etc.).
stats_t	Beinhaltet die relevanten, als Zahlen aufbereiteten Werte, die bei einem Lesezugriff auf das /proc-Dateisystem gelesen wurden.
time_vals_t	Wird zur temporären Speicherung der pro Intervall gemessenen Zeiten verwendet. Die gemessenen Zeiten werden nach Ablauf aller Messintervalle dem ensprechenden stats_t zugefügt.
stats_res_t	Dieser Datentyp beinhaltet alle Daten eines ausgewerteten Messintervalls.
stats_res_overall_t	In dieser Datenstruktur werden alle fertig ausgewerteten Daten gehalten, die zur Gestamtstatistik einer ganzen Messserie gehören.
file_coords_t	Koordinaten eines Wertes im /proc-Dateisystem. Die Koordinaten haben die Form: Liniennummer und Zeilennummer
meas_buffer_meta_t	Enthält wichtige Metainformationen, die für einen Mess-Puffer von Bedeutung sind.
meas_buffer_t	Datenstruktur eines Mess-Puffers. Ein Mess-Puffer wird verwendet, um die Rohdaten pro Intervall aus dem /proc-Dateisystem zwischen zu speichern. Die Rohdaten werden erst am Schluss einer Mess-Serie aufbereitet und ausgewertet.
file_params_t	Diese Datenstruktur enthält die Metainformationen zu jedem Wert, der aus dem /proc-Dateisystem gelesen werden soll. Dazu gehören beispielsweise der Dateipfad, die Koordinaten des Messwertes, eine Referenz auf die Rohdaten des Dateiinhalts, etc.

Tabelle 3.22: Kurzbeschrieb der meas-Datentypen

Detailansicht Module

Basierend auf der Gesamtübersicht, wurden die verschiedenen Module implementiert. Die Modulübersicht soll eine etwas detailliertere Übersicht darstellen. Bei der folgenden Illustration wurden der Übersichtlichkeit wegen pro Modul nur relevante, modulübergreifende Funktionen berücksichtigt.

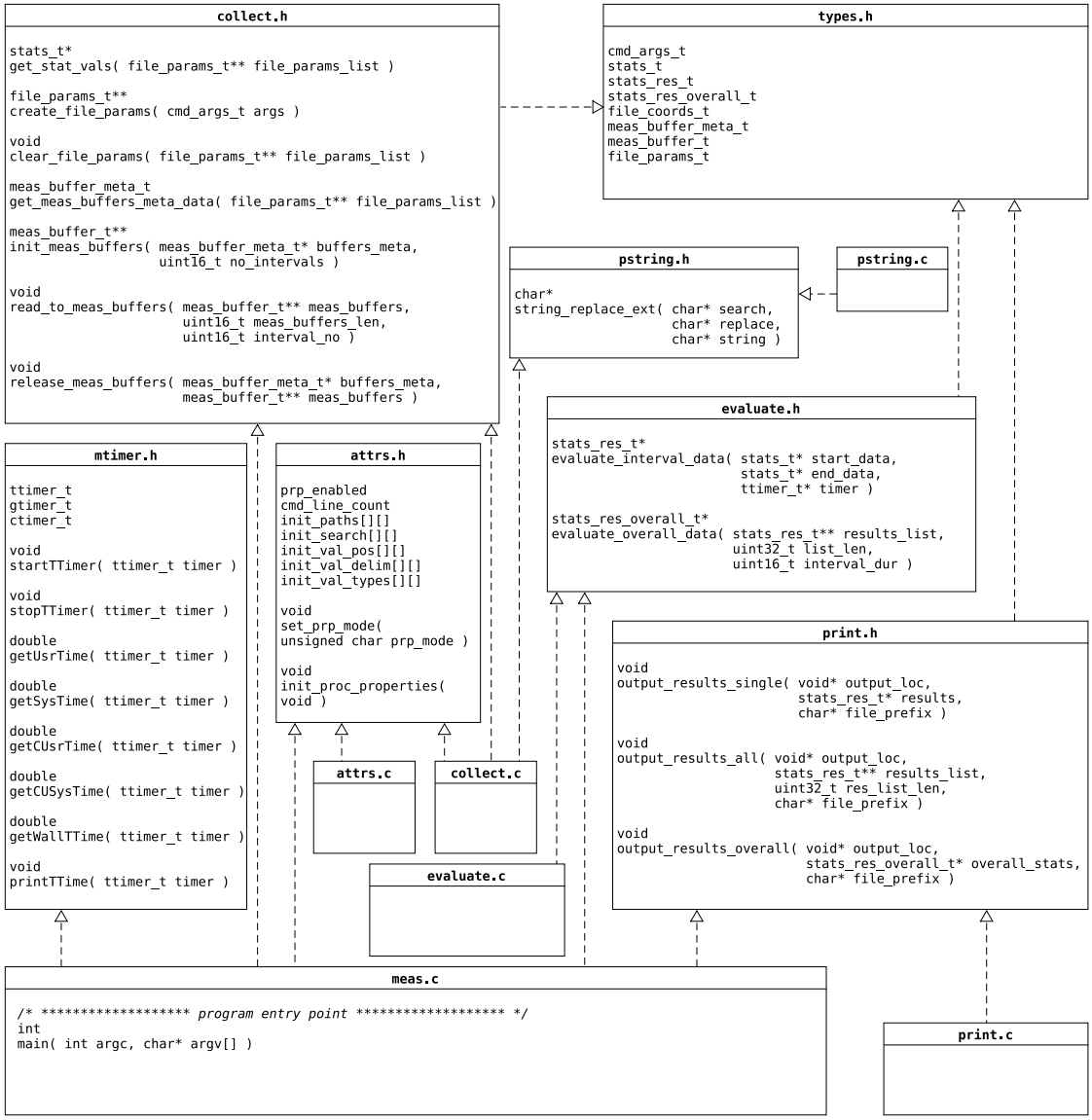


Abbildung 3.6: Detailansicht meas-Module

meas Diese Komponente ist der Programmeinstiegspunkt. Sie ist für das Einlesen Programmargumente zuständig, sowie für die Koordination des Hauptprogrammablaufs. Beispielsweise werden hier die Aufrufe zum Starten einer Messung ausgeführt, wie auch die Intervall-Wartezeiten berechnet, etc.

mtimer mtimer.h ist eine Programm-Header-Datei, die von Prof. Dr. M. Thaler (ZHAW, InIT) geschrieben wurde. Sie enthält C-Makros, die einen universellen, vereinfachten Zugriff auf verschiedene Timer-Module eines Standard-Linux-Systems ermöglichen. Für die meas-Applikation sind zwei Timer-Typen im Einsatz:

- Timer mit einer Auflösung von $10ms$ für das Messen der effektiven Intervallzeiten.

- Timer mit einer Auflösung von $1\mu s$, um die Lesezeiten aus dem /proc-Dateisystem aufzuzeichnen.

attrs Diese Komponente repräsentiert die Schnittstelle zum /proc-Dateisystem. Hier wird spezifiziert, welche Dateien ausgelesen werden und an welcher Position (Zeile/-Spalte) sich der Datenwert pro Datei befindet, welcher für die Statistikevaluation benötigt wird.

Diese Konfigurationsparameter sind in globalen Variablen gespeichert und werden beim Starten der Applikation festgelegt. Andere Module greifen während des Ablaufs einer Messserie nur lesend auf die Variablen zu.

```
1  ...
2  unsigned char prp_enabled;
3  unsigned char cmd_line_count;
4  char init_paths    [ MAX_PROC_VALS ][ PATH_LEN ];
5  char init_search   [ MAX_PROC_VALS ][ PATH_LEN ];
6  char init_val_pos  [ MAX_PROC_VALS ][ PATH_LEN ];
7  char init_val_delim[ MAX_PROC_VALS ][ PATH_LEN ];
8  char init_val_types[ MAX_PROC_VALS ][ PATH_LEN ];
9  ...
```

Listing 3.1: Deklaration der Konfigurationsfelder

prp_enabled gibt Auskunft darüber, ob die meas-Appliktion im PRP-Mode gestartet wurde oder nicht. cmd_line_count enthält die Information darüber, wie viele Werte jeweils aus dem /proc-Dateisystem gelesen werden.

Initialisierung eines Konfigurations-Datenfeldes:

```
1  ...
2  strncpy( init_paths[ INDEX_UTIME        ], "/proc/$PID$/stat", PATH_LEN );
3  strncpy( init_paths[ INDEX_STIME        ], "/proc/$PID$/stat", PATH_LEN );
4  strncpy( init_paths[ INDEX_IF_NAME_PRP  ], "/proc/net/dev"   , PATH_LEN );
5  ...
```

Listing 3.2: Beispiel eines Konfigurationsfeldes, /proc-Dateipfade

Mit $-Zeichen umgebene Zeichen im Dateipfad werden in diesem Fall durch die Prozess-ID ersetzt, die dem Programm als Argument übergeben wird.

Initialisierung eines Konfigurations-Datenfeldes:

```
1  ...
2  strncpy( init_val_pos[ INDEX_UTIME        ], "1:14", PATH_LEN );
3  strncpy( init_val_pos[ INDEX_STIME        ], "1:15", PATH_LEN );
4  strncpy( init_val_pos[ INDEX_IF_NAME_PRP  ], "7:1" , PATH_LEN );
5  strncpy( init_val_pos[ INDEX_RX_BYTES_PRP ], "7:2" , PATH_LEN );
6  strncpy( init_val_pos[ INDEX_TX_BYTES_PRP ], "7:10", PATH_LEN );
7  ...
```

Listing 3.3: Beispiel eines Konfigurationsfeldes, Datei-Koordinaten

pstring Dieses Modul behandelt das Ersetzen von einmal oder mehrfach vorkommenden Zeichenketten in einer Zeichenkette. So wird es beispielsweise möglich, das Muster «PID» in einem Dateipfad durch eine Prozess-ID zu ersetzen.

collect Dieser Programmteil ist dafür zuständig, die in «attrs» definierten Werte aus dem /proc-Dateisystem zu lesen. Besonders erwähnenswert sind die Lesepuffer. Die Initialisierung sowie die Schreib- und Leseoperationen dieser Puffer werden alle in diesem Programm-Modul abgehandelt. Des Weiteren ist «collect» dafür zuständig, die in den Puffern gesammelten Rohdaten aus /proc für die spätere Auswertung aufzubereiten. Ferner befinden sich hier administrative Funktionen, die die Datenstrukturen betreffend den /proc-Datei-Metadaten behandeln.

evaluate Dieses Modul enthält alle Funktionen, welche aus den gesammelten Messdaten statistische Werte berechnen, sowohl pro Messintervall als auch eine Gesamtstatistik über alle Messintervalle. Die berechneten Statistiken werden in den entsprechenden Datenstrukturen abgelegt und können anschliessend für Ausgaben auf bliebige Ausgabe-Geräte weiterverwendet werden.

print Die Implementationen im Print-Modul dienen dazu, Datenstrukturen mit Intervall-Resultaten oder Gesamtstatistiken auf beliebige Ausgabegeräte auszugeben. Der Umstand, dass in einem unixoiden System alles als eine Datei angesehen wird, macht es möglich, dass die Ausgabefunktionen generisch implementiert werden können. So kann den Ausgabefunktionen ein C-Pointer auf eine entsprechende Ausgabedatei übergeben werden. Ausgabedateien haben beispielsweise die Form einer Ausgabekonsole, einer einfachen Text-Datei, einer Druckerwarteschlange, etc.

Ablauf einer Messserie

Das folgende Sequenzdiagramm, verdeutlicht den Ablauf einer Messserie mit meas. Die Namen der Funktionsparameter entsprechen nicht genau denen aus dem Quellcode, um die Interpretaion des Sequenzdiagramms zu vereinfachen.

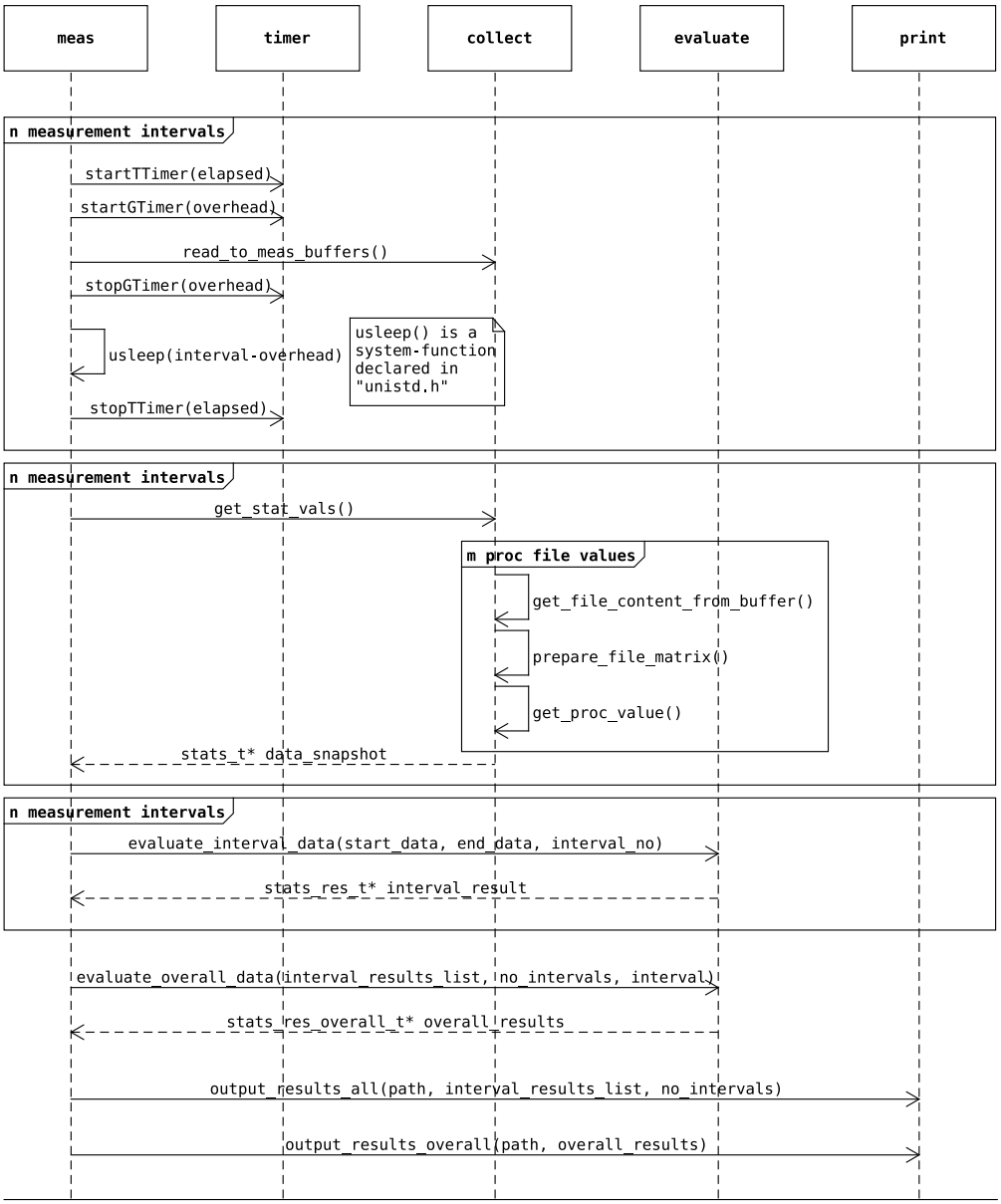


Abbildung 3.7: Ablaufdiagramm meas-Messserie

Minderung von Messfehlern / Optimierungen

Die Zeit, welche vergeht, um die Datenwerte aus dem /proc-Dateisystem zu lesen, verfälscht die Messung eines Intervalls. Dieser Zeitabschnitt sollte deshalb möglichst kurz gehalten werden. Während des Entwicklungsprozesses von meas wurden zwei Optimierungsschritte durchgeführt, welche der Verkürzung dieses Zeitabschnitts gewidmet wurden.

Der Messfehler wird pro Messintervall auf eine Genauigkeit von $1\mu s$ protokolliert und in den Ausgaben von meas angezeigt, unter der Bezeichnung «Overhead Time».

Resultate ohne Optimierungen Beim Durchführen erster Messserien mit meas, wurde ein Messfehler von $7ms$ festgestellt, was im Vergleich zur Intervallzeit einen relativ hohen Anteil darstellt.

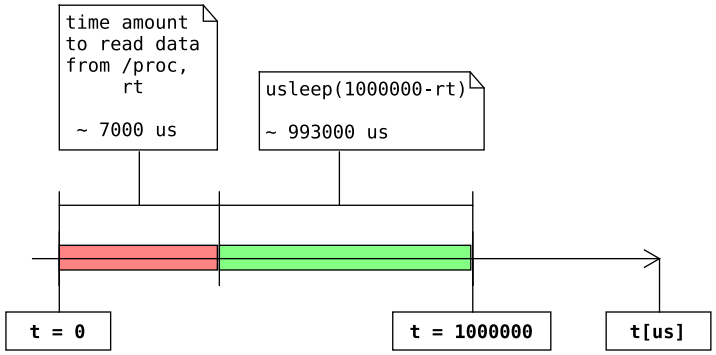


Abbildung 3.8: Messfehler ohne Optimierungen. Intervallzeit 1 s.

Resultate nach erster Optimierungsstufe Nach sorgfältiger Analyse des Quellcodes wurde festgestellt, dass das Interpretieren und Kopieren der gelesenen Werte sehr viel Zeit in Anspruch nahmen. Hauptsächlich durch zahlreiche Aufrufe der memcpy-Funktion mit nur kleinen Datenbereichen. Die memcpy-Funktion kopiert eine definierte Anzahl von Bytes von einer Quell- an eine Zieladresse im Arbeitsspeicher. Diese Aufrufe konnten signifikant reduziert werden. Nach den Code-Optimierungen war noch ein Messfehler von $600\mu s$ zu verzeichnen.

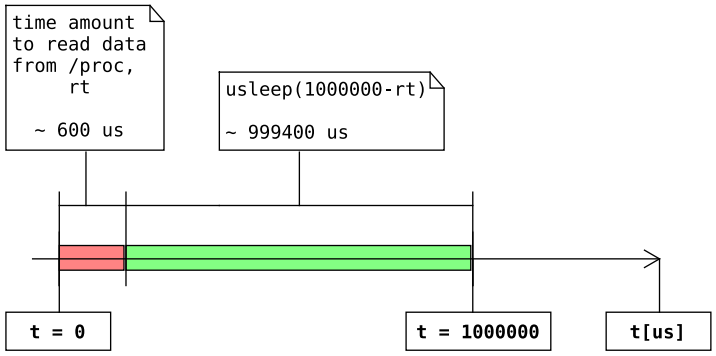


Abbildung 3.9: Messfehler nach erster Optimierungsstufe. Intervallzeit 1 s.

Resultate nach zweiter Optimierungsstufe Zusätzlich zum Auslesen der Werte wurden die Statistiken eines Intervalls bereits ausgewertet. Ebenfalls wurde je nach Konfigurationsparameter der Software pro Intervall eine Konsolenausgabe getätigt, was einen zusätzlichen Zeitverlust bedeutete.

Das Konzept, wie das Auslesen der Werte aus dem /proc-Dateisystem umgesetzt wird, wurde grundlegend geändert, zwecks weiterer Minimierung des Messfehlers, um eine

höhere Messgenauigkeit zu erreichen. Wie bereits im Sequenzdiagramm (siehe 3.2.4.1 auf Seite 45) ersichtlich ist, wurde ein Lesepuffer eingeführt, der die gelesenen Werte zwischenspeichert und die Auswertungen und Ausgaben erst nach Beenden einer Messserie durchgeführt werden. Die Funktion, welche für das Auslesen der Werte aus dem /proc-Dateisystem zuständig ist, wurde mit möglichst wenigen Instruktionen implementiert und hat folgende Charakteristik:

```
1  void read_to_meas_buffers( meas_buffer_t** meas_buffers,
2                             uint16_t meas_buffers_len,
3                             uint16_t interval_no )
4  {
5    uint16_t i;
6    ssize_t bytes_read;
7    off_t offset_location;
8    uint32_t c_buf_offset;
9    meas_buffer_t* mb = NULL;
10
11   for( i = 0; i < meas_buffers_len; i++ )
12   {
13     mb = ( *( meas_buffers + i ) );
14     c_buf_offset = mb->slice_size * interval_no;
15     bytes_read = -1;
16     offset_location = -1;
17
18     /*
19      * Read to the given file_descriptor. The start position of the
20      * file buffer is the start address of our content_buffer
21      * with an offset of our buffersize of one file read multiplied by
22      * the interval number.
23      * The last argument passed to read is the size of one file read.
24      * The the status of the file at each intervals time is stored
25      * consecutively in the content_buffer.
26      * */
27     bytes_read = read( mb->file_descriptor,
28                        ( mb->content_buffer + c_buf_offset ),
29                        mb->slice_size );
30
31     /* reposition the file contents pointer to the beginning of the
32      * file after reading from it */
33     offset_location = lseek( mb->file_descriptor, 0, SEEK_SET );
34
35     if( bytes_read == -1 || offset_location == -1 )
36     {
37       perror( "Failed to read from file. Exiting ...\n" );
38       exit( EXIT_FAILURE );
39     }
40     else if( bytes_read >= mb->slice_size )
41     {
42       perror( "File read buffer is too small, "
43               "for the requested file. Exiting" );
44       exit( EXIT_FAILURE );
45     }
46     else
47     {
48       /* NULL-Terminate the text. */
49       *( mb->content_buffer + c_buf_offset + bytes_read ) = '\0';
50     }
51   }
52 }
```

Listing 3.4: Funktionskörper, Auslesen aus dem /proc-Dateisystem

Aufgrund dieser weiteren Optimierung wurde abermals ein betrachtlicher Gewinn an Performanz erreicht. Die durchschnittliche Zeit für das Auslesen der Dateien aus dem /proc-Dateisystem beträgt nach der zweiten Optimierungsstufe noch $100\mu s$.

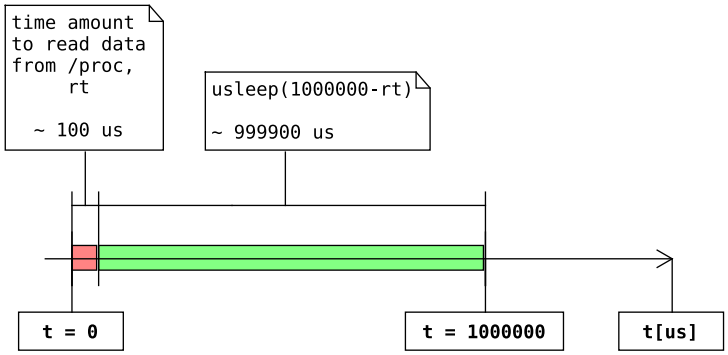


Abbildung 3.10: Messfehler nach zweiter Optimierungsstufe. Intervallzeit 1 s.

Nach diesen beiden Optimierungsstufen wurde eine im Rahmen dieser Arbeit akzeptable Zeitspanne erreicht, um die statistischen Werte auszulesen. Ferner sei erwähnt, dass auch die Aktualisierungs-Frequenz der Netzwerkstatistiken im /proc-Dateisystem auf $100\mu s$ limitiert ist, siehe 3.1.2.4 auf Seite 27. Nach diesen Optimierungen ist es nicht mehr möglich, ausgewertete Resultate direkt während der Messung auf einer Ausgabekonsole zu verfolgen. Der Fokus dieser Arbeit liegt jedoch darin, Messungen aufzuzeichnen und anschliessend die ausgewerteten Resultate zu dokumentieren.

Systemcalls und Datenkonsistenz

Um von den Dateien aus /proc zu lesen, wurden direkt die Systemcalls des Betriebssystems $open()$ [40], $read()$ [12], $lseek()$ [45] und $close()$ [11] verwendet. Beim Öffnen der Dateien ($open()$-Systemcall) werden die Flags O_RDONLY (nur Lesezugriff) und O_SYNC verwendet. O_SYNC garantiert die Datenkonsistenz. Gleichzeitige Lese- und Schreibzugriffe auf Dateien werden somit vermieden. Diese Situation könnte auftreten, wenn beispielsweise der Kernel die /proc-Statistiken aktualisiert und zur gleichen Zeit meas lesend auf die entsprechende Datei zuzugreifen versucht.

Funktionstests

Um sicherzustellen, ob die Werte korrekt ausgelesen und in entsprechende Zahlenformate umgewandelt werden, wurde ein Unit-Test geschrieben. Der Test liest Werte aus einer Datei, und vergleicht die erhaltenen Resultate mit erwarteten Werten. Wenn ein ausgelesener Wert nicht mit dem erwarteten Wert übereinstimmt, schlägt der gesamte Test fehl. Für das Vergleichen der Werte wird die $assert()$-Funktion [50] verwendet:

```
1  #define UTIME_EXPECTED             8319
2  #define STIME_EXPECTED            61330
3  #define IF_NAME_EXPECTED        "eth0:"
4  #define RX_BYTES_EXPECTED   208868176ul
5  #define TX_BYTES_EXPECTED  9437161222ul
```

```
6
7
8  int main( int argc, char* argv[] )
9  {
10   ...
11
12   file_params_list = create_file_params( args );
13
14   buffers_metadata = get_meas_buffers_meta_data( file_params_list );
15   buffers = init_meas_buffers( &buffers_metadata, ( uint16_t )args.no_intervals );
16
17   read_to_meas_buffers( buffers, buffers_metadata.unique_paths_count, 0 );
18
19   data_snapshot = get_stat_vals( file_params_list, buffers,
20                                  buffers_metadata.unique_paths_count, 0 );
21
22   release_meas_buffers( &buffers_metadata, buffers );
23
24   assert( data_snapshot->utime == UTIME_EXPECTED );
25   assert( data_snapshot->stime == STIME_EXPECTED );
26   assert( strcmp( data_snapshot->if_name_if0, IF_NAME_EXPECTED ) == 0 );
27   assert( data_snapshot->rx_bytes_if0 == RX_BYTES_EXPECTED );
28   assert( data_snapshot->tx_bytes_if0 == TX_BYTES_EXPECTED );
29
30   printf( "All unit tests successful.\n" );
31
32   ...
33 }
```

Listing 3.5: Code-Ausschnitt, meas Unit Test

Speicherüberprüfung

Bei der Implemtierung der meas-Applikation wurde das Konzept der dynamischen Speicherallozierung ($malloc()$ [52], $free()$ [51]) angewendet. Um zu überprüfen, ob jede Allokation von Speicher jeweils wieder korrekt freigegeben wurde, also keine sogenannten «Speicherlecks» entstehen, wurde die Profiling-Software «valgrind» (siehe Tabelle 3.17 auf Seite 38) eingesetzt. Nach jedem Release einer Version von meas wird ein Profiling-Rerport erstellt. Dieser hat die nachfolgend gezeigte Form:

```
==20699== Memcheck, a memory error detector
==20699== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==20699== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==20699== Command: ./meas -p 2430 -i 0.01 -n 500 -v 1 -m 1
==20699==
no of intervals: 500 Using "/tmp" as file output location.
Writing to: /tmp/meas_interval_results_1431290017
Writing to: /tmp/meas_overall_stats_1431290017
==20699==
==20699== HEAP SUMMARY:
==20699==     in use at exit: 0 bytes in 0 blocks
==20699==   total heap usage: 1,613 allocs, 1,613 frees, 12,835,336 bytes allocated
==20699==
==20699== All heap blocks were freed -- no leaks are possible
==20699==
==20699== For counts of detected and suppressed errors, rerun with: -v
==20699== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 13 from 6)
```

Plausibilitätskontrolle
Um sicherzustellen, ob die berechneten Statistiken/Resultate von meas plausible Werte aufweisen, wurden diese Resultaten anderer Tools, gegenübergestellt. In Kapitel 3.2.6 auf Seite 63 sind die Ergebnisse der besagten Gegenüberstellung detailliert beschrieben. Des Weiteren wurden Messungen durchgeführt mit maximaler Lasterzeugung bezüglich Data-Link-Layer. Diese Resultate wurden sodann mit der theoretischen Obergrenze, die eine Bitrate eines 100-MBit/s-Interface erreichen kann, verglichen, siehe Kapitel 3.3.1.3 auf Seite 78.

Ausgaben der Messresultate
Als Ausgabe von meas wird die Funktion $fprintf()$ [1] verwendet, welche eine Textausgabe auf verschiedene Ausgabegeräte ermöglicht.

Exemplarische Ausgabe eines Messintervalls in eine Textdatei:

```
*********************************************************************************************
 Process specific statistics, interval no:     0
 PID:  7327   Time: elapsed 1.000s, user 0.31s, sys 0.06s, overhead 120us
 CPU:  cpu workload 36.99%
 Machine wide network stats for PRP involved interfaces, not process specific:
 Interface       RX Bytes       TX Bytes      RX Bitrate [MBit/s]      TX Bitrate [MBit/s]
 =============================================================================================
 prp (prp1)             0       12287184                     0.00                    98.28
 if0 (eth0)           140       12339892                     0.00                    98.71
 if1 (eth1)           140       12339892                     0.00                    98.71
 =============================================================================================
*********************************************************************************************
```

Exemplarische Ausgabe eines Reports über eine gesamte Messserie in eine Textdatei:

```
*********************************************************************************************
 Native location: /results/testing/02_PRP/02.U.MAX.K
 Overall statistics of Process with PID:   7327   |   Total elapsed time [s]:       300.02
                                                  |   Total overhead time [us]:      35966
 Total no of intervals:              300
 Duration of each interval [s]:     1.000
 -------------------------------------------------------------------------------------------
 Elapsed time [s]:   Average:    1.00
 (per interval)      Min:        1.00 at interval: [  100.000s,   101.000s]
                     Max:        1.00 at interval: [  244.000s,   245.000s]
 -------------------------------------------------------------------------------------------
 Overhead time [us]: Average:     119
 (per interval)      Min:          89 at interval: [  162.000s,   163.000s]
                     Max:         232 at interval: [  100.000s,   101.000s]
 -------------------------------------------------------------------------------------------
 Usertime [s]:       Average:    0.31
                     Min:        0.25 at interval: [  264.000s,   265.000s]
                     Max:        0.34 at interval: [   78.000s,    79.000s]
 -------------------------------------------------------------------------------------------
 Systemtime [s]:     Average:    0.06
                     Min:        0.03 at interval: [    5.000s,     6.000s]
                     Max:        0.12 at interval: [  264.000s,   265.000s]
 -------------------------------------------------------------------------------------------
 CPU workload [%]:   Average:   36.71
                     Min:       35.00 at interval: [  130.000s,   131.000s]
                     Max:       38.00 at interval: [   33.000s,    34.000s]
 -------------------------------------------------------------------------------------------

 Machine wide statistics of the network interfaces:
 Following statistics are bound to a specific NIC not to the PID mentioned above.
 ===========================================================================================
 Statistics for virtual PRP interface with label: prp1
 -------------------------------------------------------------------------------------------
            RX [MBit/s] | at interval              ||| TX [MBit/s] | at interval
           -------------------------------------------------------------------------------
 Average:           0.00 |                      --- |||       98.06 |                      ---
 Min:               0.00 | [    0.000s,     1.000s] |||       97.64 | [   38.000s,    39.000s]
 Max:               0.00 | [  286.000s,   287.000s] |||       98.45 | [  114.000s,   115.000s]
           -------------------------------------------------------------------------------
 Total received bytes:              660
 Total transmitted bytes:    3677415293
 -------------------------------------------------------------------------------------------
 Statistics for first PRP involved physical interface with label: eth0
 -------------------------------------------------------------------------------------------
            RX [MBit/s] | at interval              ||| TX [MBit/s] | at interval
           -------------------------------------------------------------------------------
 Average:           0.00 |                      --- |||       98.71 |                      ---
 Min:               0.00 | [    1.000s,     2.000s] |||       98.66 | [   68.000s,    69.000s]
 Max:               0.00 | [  202.000s,   203.000s] |||       98.72 | [    9.000s,    10.000s]
           -------------------------------------------------------------------------------
 Total received bytes:            21700
 Total transmitted bytes:    3701774730
 -------------------------------------------------------------------------------------------
 Statistics for second PRP involved physical interface with label: eth1
 -------------------------------------------------------------------------------------------
            RX [MBit/s] | at interval              ||| TX [MBit/s] | at interval
           -------------------------------------------------------------------------------
 Average:           0.00 |                      --- |||       98.71 |                      ---
 Min:               0.00 | [    1.000s,     2.000s] |||       98.66 | [   68.000s,    69.000s]
 Max:               0.00 | [  202.000s,   203.000s] |||       98.73 | [  259.000s,   260.000s]
           -------------------------------------------------------------------------------
 Total received bytes:            21700
 Total transmitted bytes:    3701774730
 -------------------------------------------------------------------------------------------
 ===========================================================================================
*********************************************************************************************
```

3.2.4.2 Netzwerklast-Generierung «shck»

Für die Generierung der Netzwerklast wurde ein Python-Skript (mit dem Namen «shck», kurz für das schweizerdeutsche Verb «schick» (äquivalent zum deutschen Verb «senden»)) programmiert. Das Skript verwendet das interaktive Paket-Manipulierungs-Programm Scapy [44] Version 2.3.1 und Python Version 2.7.

Der Grund für die Erstellung von shck ist, dass kein Tool gefunden wurde, mit welchem die von den Testfällen geforderte Netzwerklast generiert werden konnte (Zum Beispiel UDP-Pakete, welche permanent mit der minimalen Grösse versendet werden oder das Versenden von reinen Ethernet-Frames mit beliebigen Grössen). Mit shck ist es möglich genau definierte Lasten zu erzeugen.

Der Quellcode ist der Arbeit beigelegt (siehe Kapitel 14.2 auf Seite 165). In diesem Abschnitt gibt es eine Einführung in die Parameter/Optionen von shck:

Parameter	Beschreibung
-d	Destination / Ziel Je nach Übertragungsart (Reine Layer-2-Frames, TCP- oder UDP-Pakete) ist hier eine MAC- oder IP-Adresse anzugeben. Diese Adresse ist dann das Ziel der zu generierenden Netzwerklast.
-f	File / Datei Datei, wessen Inhalt für die Payload der Frames / Pakete verwendet werden soll. Es wird pro Frame / Paket jeweils immer nur die ersten x Bytes der Datei verwendet.
-h	Hilfe Zeigt die englische Readme-Datei an.
-i	Interface Bestimmt, über welches Netzwerkinterface die Netzwerklast versendet werden soll.
-m	MTU Bestimmt die MTU (Standard-Wert beträgt 1500)
-p	Portnummer TCP-/UDP-Verbindung auf einem bestimmten Port herstellen (Standard: 52015). Client und Server müssen den selben Port ausgewählt haben.

Tabelle 3.23: Parameter der Applikation «shck» (1/2)

Parameter	Beschreibung
-P	PRP-Modus (benötigt kein Argument) Wird dieser Parameter mit angegeben, wird bei einer MTU von 1500 6 Bytes der Payload abgezogen, um Platz für den RCT des PRP-Protokolls garantieren zu können.
-n	Count / Anzahl Bestimmt, wie viele Frames / Pakete generiert werden. Der Standard-Wert ist 1 Frame / Paket. Wird die Anzahl 0 angegeben, so werden unendlich viele Frames / Pakete versendet.
-s	Sizetype / Grössentyp Bestimmt den Grössentyp der Netzwerklast. Zur Auswahl stehen «MIN», «MAX» und «CUSTOM». Bei «MIN» und «MAX» besteht die Netzwerklast jeweils aus lauter kleinen oder grossen Frames / Paketen. Mit «CUSTOM» als Argument erhalten die Frames / Pakete jeweils eine Grösse aus einer Liste, die Zeile für Zeile eingelesen wird. Diese Grössen werden aus einer Datei ausgelesen, die sich im selben Ordner wie shck befindet. Dabei werden automatisch Grössen, die grösser als die angegebene MTU (-m Parameter) sind, auf die maximal mögliche Grösse gesenkt. Müssen mehr Frames / Pakete als Zeilen in der Liste versendet werden, wird die Liste wieder von dessen Beginn aus abgearbeitet, sobald shck das Ende der Liste erreicht.
-S	Server-Modus Startet shck im Server-Modus, um TCP-/UDP-Pakete empfangen zu können. Wird dieser Parameter nicht mit angegeben, startet shck im Client-Modus. Benötigt den «-t»-Parameter, um den Server im TCP- oder UDP-Modus starten zu können.
-t	Transmission type / Übertragungsart Bestimmt, wie die Frames / Pakete versendet werden sollen. Zur Auswahl stehen «ETH» (Layer 2), «TCP» und «UDP». Je nach Auswahl muss das Argument zum Parameter -d definiert werden.

Tabelle 3.24: Parameter der Applikation «shck» (2/2)

shck wurde anhand von Profiling-Tools und Wireshark (Whitebox-Ansatz) untersucht, um darauf Optimierungen an der Applikation vornehmen zu können. So wurden die von shck generierten Netzwerklasten mit Wireshark überprüft, um feststellen zu können, ob die Last wie gewünscht erzeugt wird. Mit dem Python Profiler cProfile erzeugte man während der Ausführung von shck ein Profil, welches darauf mit SnakeViz (`https://jiffyclub.github.io/snakeviz/`) untersucht wurde.

Mit diesen Mitteln konnte man determinieren, wo noch Optimierungsbedarf besteht, den man darauf umsetzen konnte. Des Weiteren gab es Unterstützung von Martin Renold (Wissenschaftlicher Mitarbeiter Institute of Embedded Systems, ZHAW School of

Engineering), welcher bei der Optimierung des Python-Codes Hilfe in Form von Beratung beigesteuert hat.

Frame- / Paketaufbau
Die Farbgebung in den nachfolgenden Grafiken ist wie folgt zu interpretieren:

- Blau: Hardware-gesteuerte Fragmente eines Ethernet-Frames, auf die kein Einfluss auf Betriebssystem-Ebene möglich ist.
- Grün: Von PRP-1 stack erzeugtes Ethernet-Frame-Fragment.
- Weiss / Transparent: Inhalt der Fragmente wird über Socket oder shck erzeugt.

Es ist zu beachten, dass die IP- und TCP-Header aufgrund ihrer optionalen Felder variable Längen aufweisen können. In den nächsten Abbildungen wird jedoch von festen Grössen dieser Header ausgegangen, welche von shck in jedem Fall in dieser Form generiert werden. Diese Form wird durch die Socket-Erzeugung des Betriebssystems vorgegeben.

Bei den Wireshark-Screenshots ist zu beachten, dass Wireshark die von der NIC erzeugten Daten (Präambel, SFD, FCS und IFG) nicht sieht und nicht miteinbezieht. Um die Layer-2-Grösse des Frames zu erhalten sind jedoch nur 4 Bytes für die FCS der Framegrösse dazu zu rechnen. In den Screenshots ist des Weiteren der RCT vom PRP-1 stack grün eingefärbt. Dieser wird von Wireshark nicht als RCT erkannt, sondern entweder der Payload/Data zugewiesen oder fälschlicherweise als «VSS-Monitoring ethernet trailer» erkannt.

Frames / Pakete, die mit shck generiert werden, haben die folgenden Charakteristiken:

Ethernet-Frame

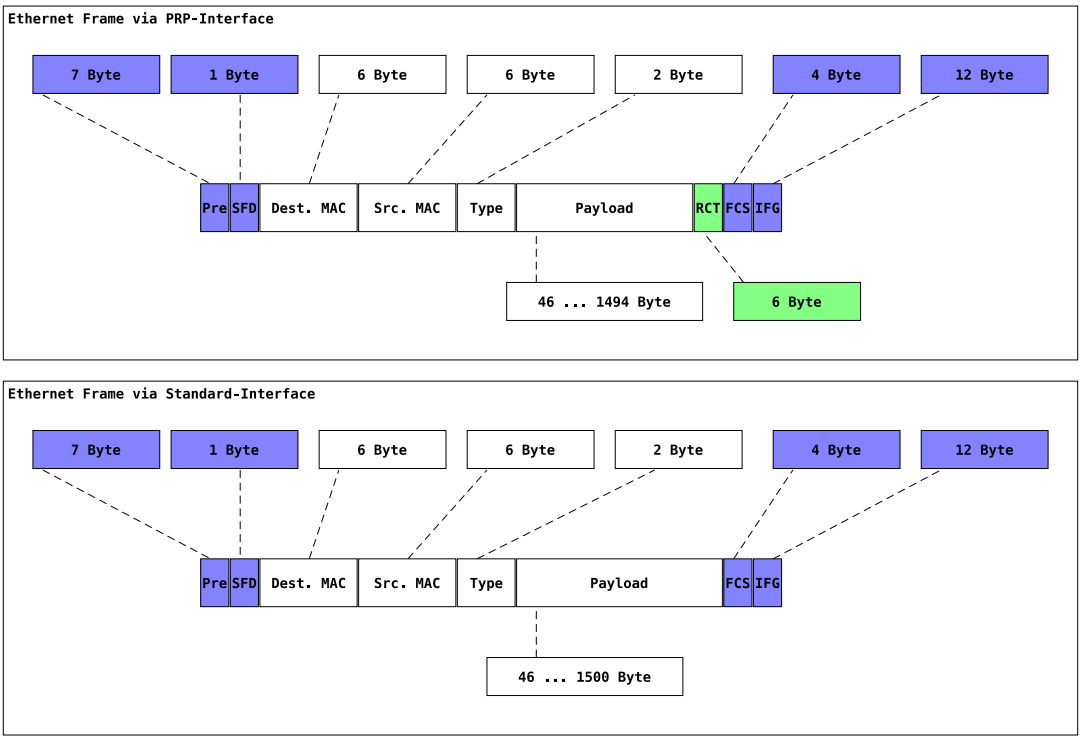


Abbildung 3.11: Ethernet-Frame, generiert mit shck via PRP- und Standard-Interface

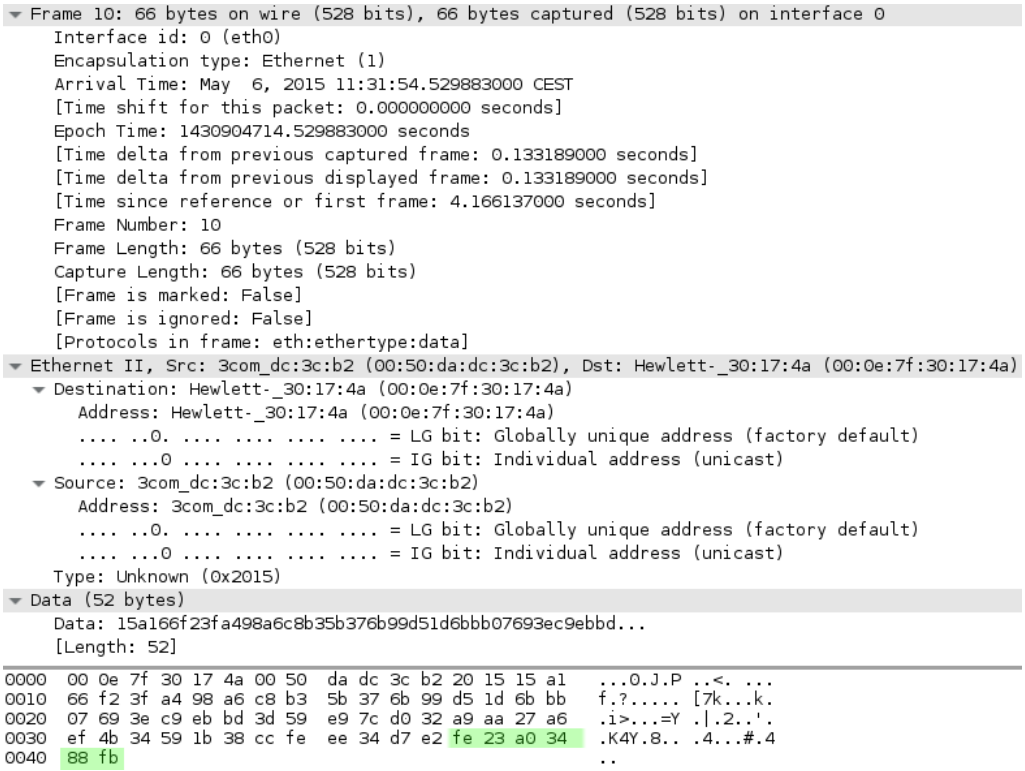
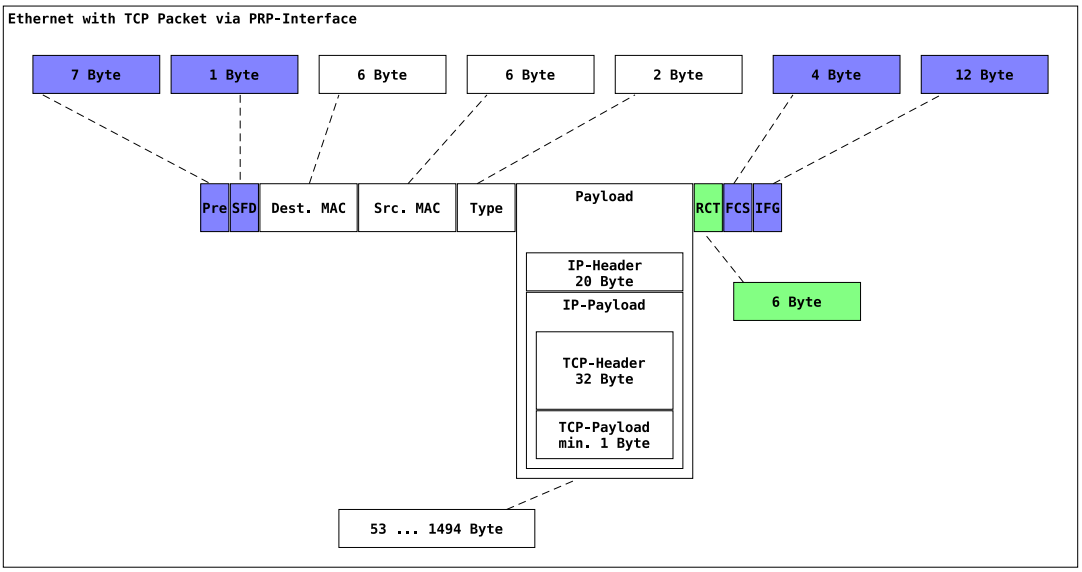


Abbildung 3.12: Wireshark: Ethernet-Frame mit minimaler Länge, versendet über Standard-Interface

TCP-Paket



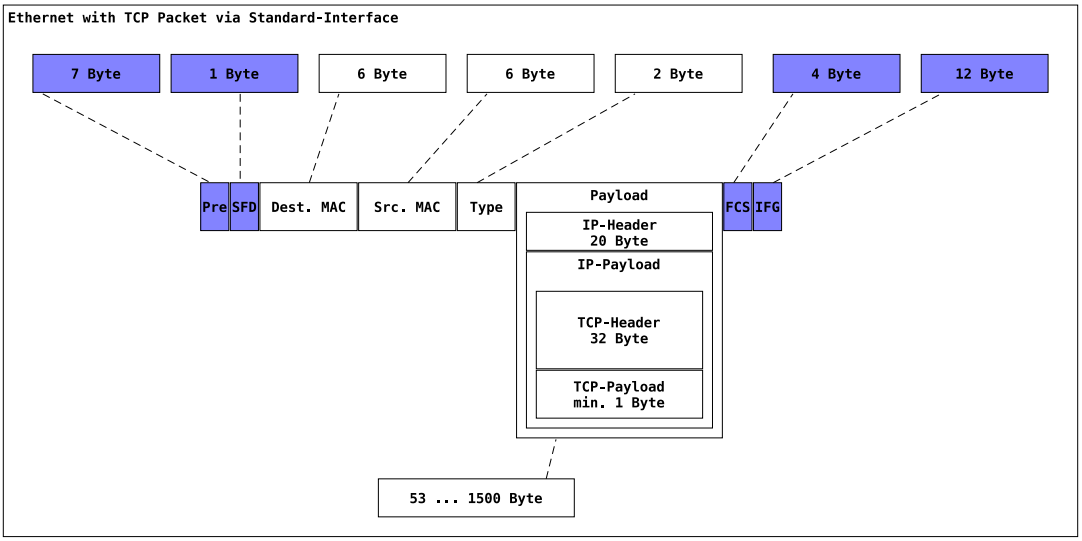


Abbildung 3.13: TCP-Paket, generiert mit shck via PRP- und Standard-Interface

```
▸ Frame 39: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
▸ Ethernet II, Src: Hewlett-_d0:17:05 (00:0f:20:d0:17:05), Dst: Hewlett-_30:17:4a (00:0e:7f:30:17:4a)
▾ Internet Protocol Version 4, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.2 (192.168.0.2)
    Version: 4
    Header Length: 20 bytes
  ▸ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 53
    Identification: 0x5180 (20864)
  ▸ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
  ▸ Header checksum: 0x67ef [validation disabled]
    Source: 192.168.0.1 (192.168.0.1)
    Destination: 192.168.0.2 (192.168.0.2)
▾ Transmission Control Protocol, Src Port: 49214 (49214), Dst Port: 52015 (52015), Seq: 1, Ack: 1, Len: 1
    Source Port: 49214 (49214)
    Destination Port: 52015 (52015)
    [Stream index: 0]
    [TCP Segment Len: 1]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 2    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header Length: 32 bytes
  ▸ .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
    Window size value: 909
    [Calculated window size: 14544]
    [Window size scaling factor: 16]
  ▸ Checksum: 0x5aac [validation disabled]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
▾ Data (1 byte)
    Data: 00
    [Length: 1]
▾ VSS-Monitoring ethernet trailer, Source Port: 65076
    Src Port: 65076

0000  00 0e 7f 30 17 4a 00 0f  20 d0 17 05 08 00 45 00   ...0.J..  .....E.
0010  00 35 51 80 40 00 40 06  67 ef c0 a8 00 01 c0 a8   .5Q.@.@. g.......
0020  00 02 c0 3e cb 2f c6 17  7f ec e4 23 3e b5 80 18   ...>./.. ...#>...
0030  03 8d 5a ac 00 00 01 01  08 0a 29 19 2e 96 29 19   ..Z..... ..)...).
0040  22 13 00 fe 34 a0 3b 88  fb                       "...4.;. .
```

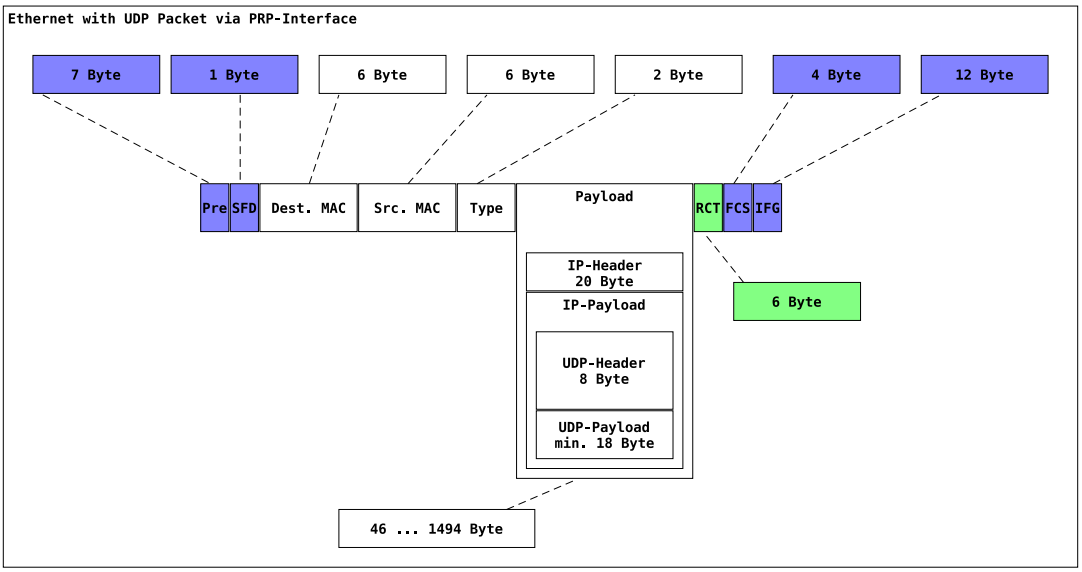
Abbildung 3.14: Wireshark: TCP-Paket mit minimaler Länge, versendet über Standard-Interface

TCP-Sockets und bytegenaue Grössen Es ist mit dem in Python erzeugten TCP-Socket nicht möglich, TCP-Pakete bytegenau zu erzeugen, da der TCP/IP-Stack teilweise die ihm übermittelten Daten gruppiert auch wenn diesem die Daten mit mehreren Send-Befehlen zugestellt werden. [38]

Aus diesem Grund besteht die Möglichkeit, dass die Frames etwas grösser ausfallen können, worunter die Reproduzierbarkeit leidet. Untersuchungen haben ergeben, dass die Netzwerklasten immer noch reproduzierbar sind, jedoch nicht mehr die exakt selben Werte. Dies kann u.a. anhand der Ergebnisse des Szenarios 01 in Kapitel 4.1 auf Seite 104 festgestellt werden.

Eine Möglichkeit, bytegenaue Grössen zu erzielen, wäre, vor jedem Send-Befehl den TCP-Socket zu öffnen und dann wieder zu schliessen, was aber kein realistischer Vorgang wäre, da übliche Kommunikationen via TCP sich in dieser Weise nicht charakterisieren. Eine TCP-Verbindung wird erst geschlossen, wenn der Sender alles übermittelt hat, was er in diesem Moment übertragen wollte.

UDP-Paket



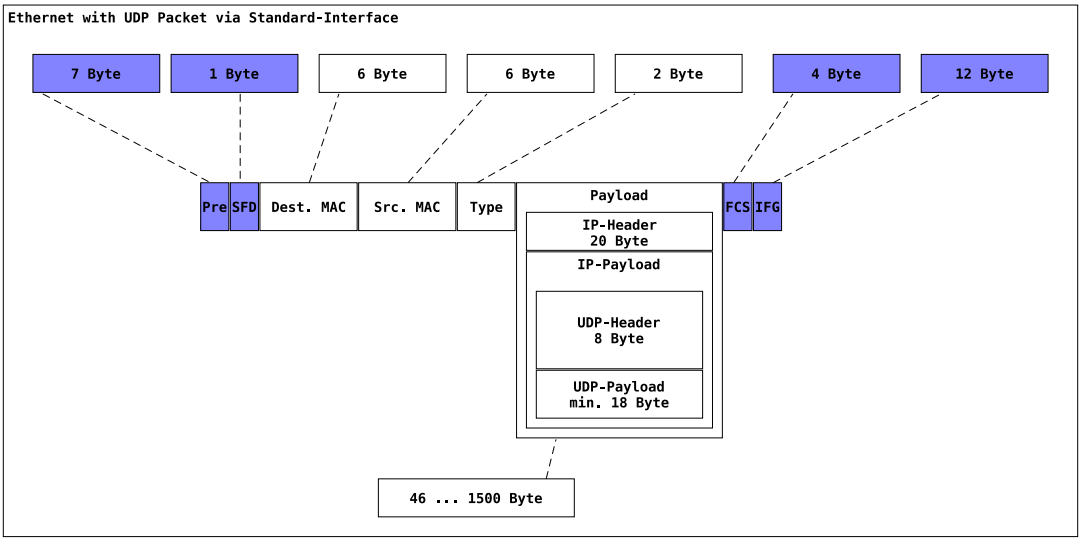


Abbildung 3.15: UDP-Paket, generiert mit shck via PRP- und Standard-Interface

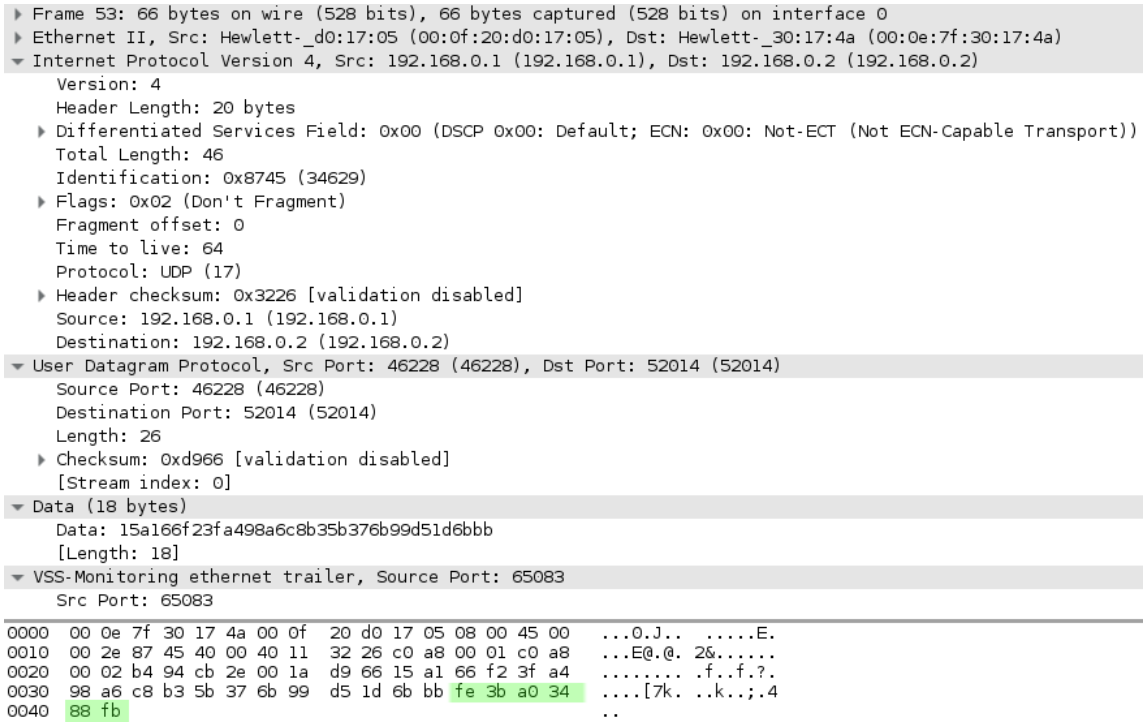


Abbildung 3.16: Wireshark: UDP-Paket mit minimaler Länge, versendet über Standard-Interface

Programm-Ablauf

shck läuft je nach angegebenen Parametern unterschiedlich ab. Die Parameter, die dies bewirken, sind in der nächsten Grafik aufgeführt. Die exakten Frame-/Paketgrössen sind dem Kapitel 3.3.1.2 auf Seite 74 zu entnehmen.

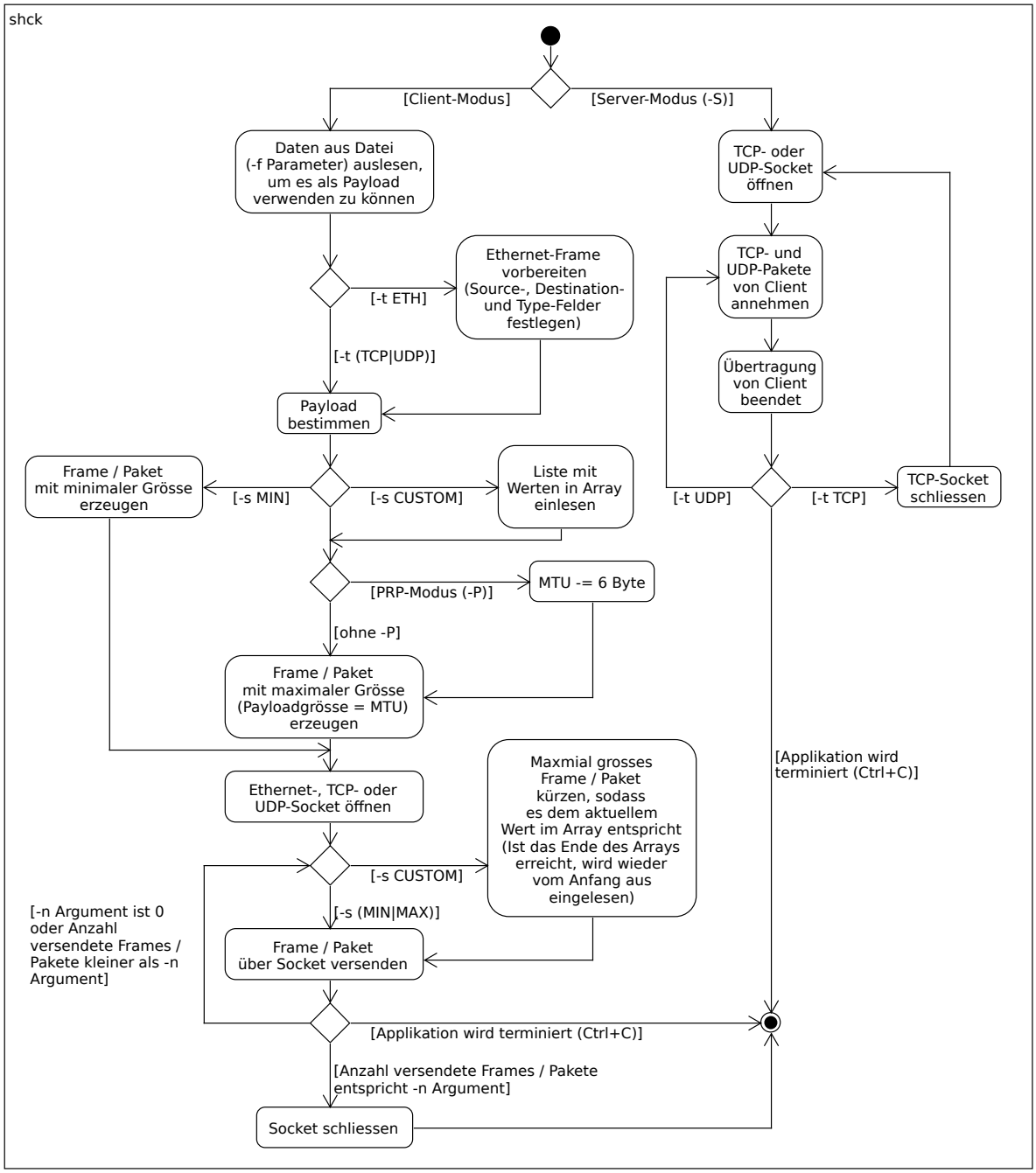


Abbildung 3.17: shck: Ablauf

3.2.5 Verwendete Tools

In diesem Kapitel werden die in dieser Arbeit verwendeten Tools aufgelistet, die es durch die engere Auswahl geschafft haben. Dazu wird aufgeführt welche Aufgabe von welcher Applikation übernommen wird.

Tool	Version	Aufgabe in dieser Arbeit
arping	2.11	Überprüfen der Erreichbarkeit der Computer im Netzwerk auf Layer 2 mittels ARP
bwm-ng	0.6	Analyse der Netzwerkbelastung
Calnex Paragon	keine	Erzeugen von Verzögerungen, Paketverlust oder anderen Störungen im Netzwerk
ECI	keine	Erzeugen von Störungen im Netzwerk
iperf3	3.0.11	Generieren und Versenden von Netzwerktraffic über TCP und UDP
meas	0.1.8	Performance-Ermittlung des PRP-1 Stacks
pidstat (sysstat)	10.0.5	Analyse des PRP-1 Stacks (siehe Kapitel 2.1.1 auf Seite 17) im Betrieb
pmap	procps-ng 3.3.3	Festellen des Speichers, der vom PRP-1 Stack beansprucht wird
shck	0.8.3	Netzwerklast-Generierung für alle Testfälle
strace	4.5.20	Anzeigen von System-Calls des PRP-1 Stacks, um dessen Verhalten nach zu vollziehen
top	procps-ng 3.3.3	Performance des PRP-1 Stacks bezüglich CPU und RAM überwachen
tshark	1.8.2	Festhalten und Analysieren des Netzwerkverkehrs
valgrind	3.10.1	Fehlersuche in eigens entwickelten C-Applikationen

Tabelle 3.25: Verwendete Tools und ihre Aufgaben in dieser Arbeit

3.2.6 Verifizierung der Messwerte

Die Messwerte der Tools werden verifiziert, indem Messungen mit unterschiedlichen Tools gleichzeitig ausgeführt werden. Dies hat zur Folge, dass jedes Tool das exakt gleiche Verhalten misst. Die Bedingungen werden in den folgenden Kapiteln beschrieben.

Diese Verifizierung ist als Blackbox-Ansatz zu verstehen, bei dem die Ergebnisse unterschiedlicher Applikationen einander gegenübergestellt werden.

3.2.6.1 CPU

Bei der Berechnung der CPU-Werte handelt es sich lediglich um Single-Threaded-Prozesse, da der PRP-1 stack und die Tools zur Generierung der Netzwerklast single-threaded

sind. meas behandelt aus diesem und aus Zeitgründen keine Multi-Threaded-Applikationen.

Auf einem der DANs wird iperf3 (siehe Tabelle 3.5 auf Seite 32) als Server und auf einem weiteren DAN als Client ausgeführt. Es wird mit unterschiedlichen Tools gemessen, was der iperf3-Client von der CPU beansprucht. Dazu wird ein zweiter Fall untersucht, in welchem mit dem Befehl dd eine 10GB grosse Datei erstellt wird, die aus zufälligen Zahlen besteht. Danach werden die Resultate verglichen, um aufzuzeigen, welches Tool glaubwürdige Messwerte liefert.

Auf dem DAN «srv02» wird der Server mit dem Befehl `iperf3 -s` ausgeführt, während der Client auf «srv01» mit dem Befehl `iperf3 -c 192.168.0.2 -d -n 200M` gestartet wird. Mit dem Client werden hier 200 Megabytes an «srv02» via TCP übertragen (dabei wird der Client als Deamon ausgeführt). Die Datei mit zufälligen Werten wird mit dem Befehl `dd if=/dev/urandom of=/tmp/randomfile bs=1M count=10000` erstellt. Während der Übertragung und File-Generierung wird mit den folgenden Applikationen die beanspruchte CPU-Leistung gemessen:

- pidstat
- top

Eine Messung wird jeweils 5 und 10 Sekunden nach dem Start des Clients / der File-Generierung durchgeführt. Konkret werden also die Intervalle [0s, 5s] und [5s, 10s] gemessen. Dabei misst jede Applikation pro Fall (dd und iperf3) zur gleichen Zeit wie die andere die Performance-Werte.

Für jede Applikation wurde jeweils ein Bash-Script erstellt, bei dem die Ausgabe in eine Datei weitergeleitet wird. Das erste der folgenden Skripts wird ausgeführt, welches dd und iperf3 nacheinander startet und jeweils die Messungen gleichzeitig ausführen lässt.

```
1  #!/bin/sh
2  (dd if=/dev/urandom of=/tmp/randomfile bs=1M count=10000 &)
3  pid_dd=$(ps -ef | grep -w dd | cut -d' ' -f6 | head -n1)
4  (./meas.sh dd $pid_dd &)
5  (./pidstat.sh dd &)
6  (./top.sh dd $pid_dd &)
7  sleep 30
8  pkill dd
9  rm /tmp/randomfile
10 sleep 10
11 (iperf3 -c 192.168.0.2 -d -n 200M &)
12 (./meas.sh iperf3 &)
13 (./pidstat.sh iperf3 &)
14 (./top.sh iperf3 &)
```

Listing 3.6: Bash-Script zur Verifizierung der Messwerte der CPU-Messungen (cpu/00_cpu_reliability_test.sh)

```
1  #!/bin/sh
2  if [ -z $1 ]
3  then
4          echo "parameter dd or iperf3 needed"
5          exit
6  fi
7  case $1 in
```

```
8           "dd")
9                   if [ -z $2 ]
10                  then
11                          echo "pid of dd for top missing"
12                          exit
13                  fi
14                  meas -i 5 -n 3 -p $2 -v 0 -P /home/scripts/measurement/test_reliability/cpu/
                        results/dd
15                  ;;
16          "iperf3")
17                  meas -i 5 -n 3 -p $(pgrep iperf3) -v 0 -P /home/scripts/measurement/
                        test_reliability/cpu/results/iperf3
18                  ;;
19          *)
20                  echo "only use dd or iperf3 as parameter"
21                  ;;
22  esac
```

Listing 3.7: Bash-Script zur Verifizierung der Messwerte der CPU-Messungen von pidstat (cpu/meas.sh)

```
1   #!/bin/sh
2   if [ -z $1 ]
3   then
4           echo "parameter dd or iperf3 needed"
5           exit
6   fi
7
8   case $1 in
9           "dd")
10                  pidstat -u -C "dd" -h 5 2 > results/dd/pidstat_dd.txt
11                  ;;
12          "iperf3")
13                  pidstat -u -C "iperf3" -h 5 2 > results/iperf3/pidstat_iperf3.txt
14                 ;;
15          *)
16                  echo "only use dd or iperf3 as parameter"
17                  ;;
18  esac
```

Listing 3.8: Bash-Script zur Verifizierung der Messwerte der CPU-Messungen von pidstat (cpu/pidstat.sh)

```
1   #!/bin/sh
2   if [ -z $1 ]
3   then
4           echo "parameter dd or iperf3 needed"
5           exit
6   fi
7
8   case $1 in
9           "dd")
10                  if [ -z $2 ]
11                  then
12                          echo "pid of dd for top missing"
13                          exit
14                  fi
15                  top -b -n 3 -d 5.0 -p $2 > results/dd/top_dd.txt
16                  ;;
17          "iperf3")
18          top -b -n 3 -d 5.0 -p $(pgrep iperf3) > results/iperf3/top_iperf3.txt
19                  ;;
20          *)
21                  echo "only use dd or iperf3 as parameter"
```

```
22                      ;;
23  esac
```

Listing 3.9: Bash-Script zur Verifizierung der Messwerte der CPU-Messungen von top (cpu/top.sh)

Folgende Werte werden von pidstat, top und meas ausgegeben:

Tool	**%CPU [0s, 5s]**	**%CPU [5s, 10s]**	**System-time [s] [0s, 5s]**	**System-time [s] [5s, 10s]**	**User-time [s] [0s, 5s]**	**User-time [s] [5s, 10s]**
meas-Messung von dd	100.00	99.80	5.00	4.99	0.00	0.00
pidstat-Messung von dd	99.60	99.80	99.60% von 5s 4.98	99.80% von 5s 4.99	0.00	0.00
top-Messung von dd	99.90	99.90	keine Angabe	keine Angabe	keine Angabe	keine Angabe
meas-Messung von iperf3	3.40	3.20	0.16	0.15	0.01	0.01
pidstat-Messung von iperf3	3.78	3.40	3.58% von 5s 0.179	3.20% von 5s 0.16	0.20% von 5s 0.01	0.20% von 5s 0.01
top-Messung von iperf3	3.20	3.20	keine Angabe	keine Angabe	keine Angabe	keine Angabe

Tabelle 3.26: Verifizierung der Messwerte der CPU-Messungen von meas, pidstat und top

Da top keine Angaben über System- und Usertime für einen spezifischen Prozess liefert, kann über diese Werte keine Angabe gemacht werden.

Anhand der Resultate und der Tatsache, dass alle Applikationen das /proc-Dateisystem nutzen, kann davon ausgegangen werden, dass die Resultate als zuverlässig zu betrachten sind. Abweichungen lassen sich dadurch erklären, dass die Messzeitpunkte bei den Programmen nicht zur exakt selben Zeit stattgefunden haben und die Berechnungen von Software zu Software marginal unterschiedlich sein können. Was jedoch aus Zeitgründen nicht belegt werden kann ist, wie pidstat und top weils die Werte anhand der Informationen aus dem /proc-Dateisystem berechnen, weshalb nicht ausgeschlossen werden kann, dass sich die Werte aufgrund unterschiedlicher Rechnungswege unterscheiden können.

Da es sich bei meas um die eigens entwickelte Applikation handelt und dort die konkreten Rechnungswege vorliegen, wird diese für die Ermittlungen der prozessorbezogenen Werte in dieser Arbeit verwendet.

3.2.6.2 Arbeitsspeicher

Wie im vorherigen Kapitel (siehe Kapitel 3.2.6.1 auf Seite 63) wird wieder iperf3 mit den selben Parametern ein Mal als Server und ein Mal als Client auf unterschiedlichen DANs ausgeführt und beim Client gemessen. Der zweite Fall mit dd wird zudem wieder betrachtet. Während der Übertragung und Dateigenerierung wird mit den folgenden Applikationen die Arbeitsspeichernutzung gemessen:

- pidstat
- pmap
- top

Eine Messung wird jeweils 5 und 10 Sekunden nach dem Start des Clients durchgeführt. Die Messungen werden pro Fall alle zur gleichen Zeit gestartet.

Für jede Applikation wurde jeweils ein Bash-Script erstellt, bei dem die Ausgabe in eine Datei weitergeleitet wird. Das erste der folgenden Skripts wird ausgeführt, welches dd und iperf3 nacheinander startet und jeweils die Messungen gleichzeitig ausführen lässt.

```
#!/bin/sh
(dd if=/dev/urandom of=/tmp/randomfile bs=1M count=10000 &)
pid_dd=$(ps -ef | grep -w dd | cut -d' ' -f6 | head -n1)
(./pidstat.sh dd &)
(./pmap.sh dd &)
(./top.sh dd $pid_dd &)
sleep 30
sudo pkill dd
rm /tmp/randomfile
sleep 10
(iperf3 -c 192.168.0.2 -d -n 200M &)
(./pidstat.sh iperf3 &)
(./pmap.sh iperf3 &)
(./top.sh iperf3 &)
```

Listing 3.10: Bash-Script zur Verifizierung der Messwerte der Arbeitsspeicher-Messungen (ram/00_ram_reliability_test.sh)

```
#!/bin/sh
if [ -z $1 ]
then
        echo "parameter dd or iperf3 needed"
        exit
fi

case $1 in
        "dd")
                pidstat -r -C "dd" -h 5 2 > results/dd/pidstat_dd.txt
                ;;
        "iperf3")
                pidstat -r -C "iperf3" -h 5 2 > results/iperf3/pidstat_iperf3.txt
                ;;
        *)
                echo "only use dd or iperf3 as parameter"
                ;;
esac
```

Listing 3.11: Bash-Script zur Verifizierung der Messwerte der Arbeitsspeicher-Messungen von pidstat (ram/pidstat.sh)

```
#!/bin/sh
if [ -z $1 ]
then
        echo "parameter dd or iperf3 needed"
        exit
fi

case $1 in
        "dd")
       sleep 5
       pmap -x $(pgrep dd) > results/dd/pmap_dd.txt
       sleep 5
       pmap -x $(pgrep dd) >> results/dd/pmap_dd.txt
       ;;
     "iperf3")
       sleep 5
       pmap -x $(pgrep iperf3) > results/iperf3/pmap_iperf3.txt
       sleep 5
       pmap -x $(pgrep iperf3) >> results/iperf3/pmap_iperf3.txt
       ;;
     *)
       echo "only use dd or iperf3 as parameter"
            ;;
esac
```

Listing 3.12: Bash-Script zur Verifizierung der Messwerte der Arbeitsspeicher-Messungen von pmap (ram/pmap.sh)

```
#!/bin/sh
if [ -z $1 ]
then
        echo "parameter dd or iperf3 needed"
        exit
fi

case $1 in
        "dd")
                if [ -z $2 ]
                then
                        echo "pid of dd for top missing"
                        exit
                fi
                top -b -n 3 -d 5.0 -p $(pgrep dd) > results/dd/top_dd.txt
                ;;
        "iperf3")
        top -b -n 3 -d 5.0 -p $(pgrep iperf3) > results/iperf3/top_iperf3.txt
                ;;
        *)
                echo "only use dd or iperf3 as parameter"
                ;;
esac
```

Listing 3.13: Bash-Script zur Verifizierung der Messwerte der Arbeitsspeicher-Messungen von top (ram/top.sh)

In dieser Arbeit werden die Werte VSZ (Virtual Memory Size) und RSS (Resident Set Size) betrachtet, um feststellen zu können, wie viel Speicher der Prozess allgemein und wie viel er davon im Arbeitsspeicher benötigt. [32]

Folgende Werte werden von pidstat, pmap und top ausgegeben:

Tool	VSZ in Kilobytes [0s, 5s]	VSZ in Kilobytes [5s, 10s]	RSS in Kilobytes [0s, 5s]	RSS in Kilobytes [5s, 10s]
pidstat-Messung von dd	4572	4572	1632	1632
top-Messung von dd	4572	4572	1632	1632
pidstat-Messung von iperf3	2084	2084	712	712
top-Messung von iperf3	2084	2084	712	712

Tabelle 3.27: Verifizierung der Messwerte der Arbeitsspeicher-Messungen von pidstat und top

pmap gibt keine konkreten VSZ- / RSS-Werte zurück, jedoch haben die Autoren dieser Arbeit den Grund zur Annahme, dass aus den Ausgabewerten von pmap die VSZ- und RSS-Werte berechnet werden können, weil pmap die Werte direkt aus dem Pseudo-File-System /proc ausliest, in welchem die Werte vom Kernel direkt aufgeführt werden. Aus Prioritätsgründen wird auf die Berechnung anhand der Werte von pmap in dieser Arbeit nicht weiter eingegangen.

Aus den Ergebnissen von pidstat und top lässt sich sagen, dass sich die Arbeitsspeichernutzung von iperf3 innerhalb von 5 Sekunden während konstantem Senden nicht verändert. Des Weiteren werden die Ausgabewerte von pidstat und top aufgrund ihrer Übereinstimmung als zuverlässig eingestuft.

Da top weiter verbreitet ist als pidstat, wird für die Messung der Arbeitsspeichernutzung top verwendet.

3.2.6.3 Netzwerkbelastung

Im Kapitel zur Verifizierung der Messwerte der CPU-Messungen (siehe Kapitel 3.2.6.1 auf Seite 63) wird eine eigens entwickelte Applikation «meas» erwähnt, mit der die CPU-Last berechnet wird. In der gleichen Applikation wurde die Möglichkeit implementiert, die Belastung des Netzwerks zu messen. Wird anhand von iperf3 Netzwerkverkehr generiert, so werden auch von diesem Tool Angaben zur Netzwerkbelastung aufgeführt. Des Weiteren wird die Netzwerkbelastung mit bwm-ng gemessen, womit man einen weiteren Vergleichswert erhält.

In diesem Abschnitt wird überprüft, ob diese Angaben als zuverlässig betrachtet werden können. Dazu wird iperf3 mit den selben Parametern wie in Kapitel 3.2.6.1 auf Seite 63 ausgeführt, jedoch erfolgt hier die Messung zum gleichen Zeitpunkt, an dem der iperf3-Client gestartet wird.

Als zweiter Fall wird für die Generierung der Netzwerklast die eigens entwickelte Applikation shck (siehe Kapitel 3.2.4.2 auf Seite 53) verwendet, welche ähnlich wie iperf3 in

diesem Kapitel aufgerufen wird: Auf dem DAN «srv02» wird shck als Server aufgerufen, während auf «srv01» shck als Client aufgerufen wird, welcher auch via TCP 200'000 Pakete mit maximaler Grösse an den Server sendet (Als Payload dient eine Datei mit zufällig erzeugten Werten).

Betrachtet werden in diesem Kapitel die Werte, die vom PRP-Netzwerkinterface des Clients «srv01» stammen.

Der Interval, in dem in diesem Abschnitt gemessen wird, beträgt 1 Sekunde.

Das Bash-Skript, das auf «srv02» ausgeführt wird, sieht wie folgt aus:

```
1  #!/bin/sh
2  case $(hostname) in
3      srv01)
4          (iperf3 -c 192.168.0.2 -d -n 200M > /home/scripts/measurement/test_reliability/net/
               results/srv01_iperf3/iperf3_output.txt &)
5          (/home/scripts/bin/meas -i 1 -n 10 -p $(pgrep iperf3) -v 0 -P /home/scripts/measurement
               /test_reliability/net/results/srv01_iperf3 &)
6          (bwm-ng -t 1000 -c 10 -o csv -F /home/scripts/measurement/test_reliability/net/results/
               srv01_iperf3/bwm.csv -I prp1 &)
7          sleep 30
8          sleep 10
9          (/home/scripts/bin/shck -d 192.168.0.2 -f /home/scripts/shck/testpayload -n 200000 -P -
               s MAX -t TCP &)
10         (/home/scripts/bin/meas -i 1 -n 10 -p $(pgrep shck) -v 0 -P /home/scripts/measurement/
               test_reliability/net/results/srv01_shck &)
11         (bwm-ng -t 1000 -c 10 -o csv -F /home/scripts/measurement/test_reliability/net/results/
               srv01_shck/bwm.csv -I prp1 &)
12         sleep 30
13         ;;
14     srv02)
15         (iperf3 -s &)
16         (ssh root@192.168.99.1 /home/scripts/measurement/test_reliability/net/00
               _net_reliability_test.sh &)
17         sleep 30
18         sudo pkill iperf3
19         sleep 10
20         (/home/scripts/bin/shck -S -t TCP &)
21         sleep 30
22         sudo pkill shck
23         ;;
24     *)
25         echo "Error!"
26         ;;
27 esac
```

Listing 3.14: Bash-Script zur Verifizierung der Messwerte der Netzwerkbelastungs-Messungen (net/00_ram_reliability_test.sh)

Darin startet «srv02» auf «srv01» über das nicht-PRP-Netzwerk («Netzwerk Ext») über SSH die Clients sobald der Server bereit ist. Somit werden erst Daten übertragen wenn man auch bereit ist, diese zu empfangen.

	srv01 (iperf3-Client, tätigt Upload)					
Interval [s]	**bwm-ng TX Daten [MBytes]**	**bwm-ng TX-Rate [MBits/s]**	**iperf3 TX Daten [MBytes]**	**iperf3 TX-Rate [MBits/s]**	**meas TX Daten [MBytes]**	**meas TX-Rate [MBits/s]**
0-1	11.5	92.2	11.6	97.1	12.3	98.0
1-2	12.3	98.1	11.2	94.0	12.3	98.0
2-3	12.3	98.1	11.2	93.8	12.3	98.1
3-4	12.3	98.1	11.1	93.4	12.3	98.1
4-5	12.3	98.1	11.3	94.6	12.3	98.1
5-6	12.3	98.1	11.1	93.3	12.3	98.1
6-7	12.3	98.1	11.2	93.6	12.3	98.1
7-8	12.3	98.1	11.2	93.9	12.3	98.1
8-9	12.3	98.1	11.1	93.1	12.3	98.1
9-10	12.3	98.1	11.2	94.2	12.3	98.1

Tabelle 3.28: Verifizierung der Messwerte der Netzwerkbelastungs-Messungen: Werte von iperf3-Client auf «srv01», bwm-ng und meas

Die Werte von bwm-ng / meas konnten auf die Werte von iperf3 über folgende Rechnungswege zurückgerechnet werden:

$$Bitrate_{iperf3} = Bitrate_{meas} * \frac{Payload\,von\,TCP-Paket}{Gesamtgrösse\,Paket}$$

$$Bitrate_{iperf3} = 98.1\,MBytes * \frac{1442\,Bytes}{1508\,Bytes} = 93.8\,MBytes$$

$$Daten_{iperf3} = ((Daten_{meas} * \frac{Payload\,von\,TCP-Paket}{Gesamtgrösse\,Paket})/1024^2) * 1'000'000$$

$$Daten_{iperf3} = ((12.3\,MBytes * \frac{1442\,Bytes}{1508\,Bytes})/1024^2) * 1'000'000 = 11.2\,MBytes$$

Als Annahme zu diesem Rechnungsweg diente, dass iperf3 lediglich die versendete Payload betrachtet, bzw. dessen Werte auf Layer 4 anstelle Layer 2 bezieht und die Datenmenge fälschlicherweise mit Faktor 1024 berechnet.

Die Werte für die Payload des TCP-Pakets und die Gesamtgrösse des Pakets stammen aus einer Wireshark-Aufzeichnung von iperf3.

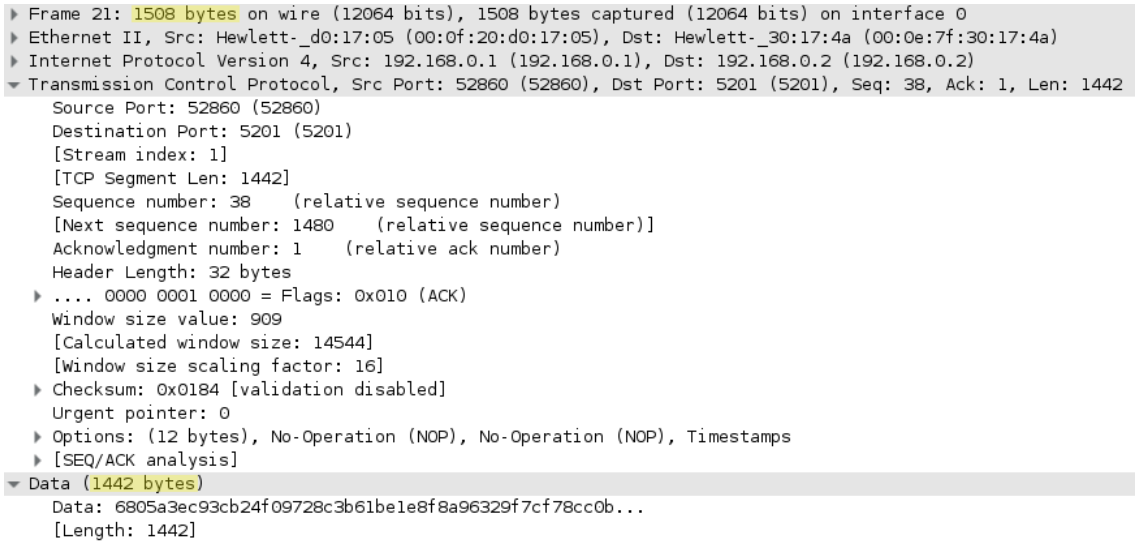


Abbildung 3.18: Wireshark-Aufzeichnung: TCP-Paket von iperf3-Client auf PRP-Interface

	srv01 (shck-Client, tätigt Upload)			
Interval [s]	**bwm-ng TX Daten [MBytes]**	**bwm-ng TX-Rate [MBits/s]**	**meas TX Daten [MBytes]**	**meas TX-Rate [MBits/s]**
0-1	7.6	60.9	8.5	68.3
1-2	12.3	98.1	12.3	98.1
2-3	12.3	98.1	12.3	98.1
3-4	12.3	98.1	12.3	98.1
4-5	12.3	98.1	12.3	98.1
5-6	12.3	98.1	12.3	98.1
6-7	12.3	98.1	12.3	98.1
7-8	12.3	98.1	12.3	98.0
8-9	12.3	98.1	12.3	98.1
9-10	12.3	98.1	12.3	98.1

Tabelle 3.29: Verifizierung der Messwerte der Netzwerkbelastungs-Messungen: Werte von bwm-ng und meas des shck-Clients auf «srv01»

Da die Resultate der unterschiedlichen Applikationen keine grossen Abweichungen zueinander haben, sind die Resultate der Applikationen als zuverlässig einzustufen. Grund für die Abweichungen könnte sein, dass die Tools wahrscheinlich nicht immer zur exakt selben Zeit messen oder in jeder Applikation bei der Berechnung anders vorgegangen wird. Wie genau bwm-ng und iperf3 bei den Berechnungen vorgehen kann aus Zeitgründen nicht ermittelt werden, jedoch ist der Rechenweg der meas-Applikation komplett ersichtlich, da es sich hierbei und eine eigens entwickelte Software handelt.

Daraus, dass bei meas das Vorgehen bei der Berechnung selbst implementiert wurde und die Werte grösstenteils mit den Werten anderer Tools übereinstimmen, wird für die Messung der Netzwerkbandbreite meas verwendet. Ein weiterer Grund, der für meas spricht, ist dessen kleiner Messfehler von circa 100us, da meas während der Messung nur aufzeichnet und am Ende die Daten auswertet und ausgibt, während z.B. bwm-ng pro Messinterval die Daten auswertet und in die Standardausgabe oder eine Datei schreibt.

3.3 Ermittlung der Performance

In diesem Kapitel wird erläutert, welche Aspekte mit welchen Mitteln und welcher Konfiguration getestet werden. Die Resultate der Tests werden in Kapitel 4 auf Seite 104 aufgeführt und analysiert.

3.3.1 Grundsätze und Rahmenbedingungen

Für die Ermittlung und Analyse der Performance werden folgende Grundsätze und Rahmenbedingungen festgelegt:

- Die ausgeführten Tests müssen reproduzier- und nachvollziehbar sein. Falls Zufallswerte verwendet werden, werden diese im Voraus generiert, gespeichert und für alle betroffenen Tests verwendet.
- Änderungen an der bestehenden Konfiguration sind vollständig dokumentiert. Ist bei einem Test keine Änderung an der Konfiguration aufgeführt, ist vom Ursprungszustand (siehe Kapitel 3.1 auf Seite 23) auszugehen.
- Wenn ein Test in verschiedenen Variationen ausgeführt wird, wird jeder dieser Tests mit den selben Mitteln durchgeführt und überprüft.
- Wird bei einem Test eigens entwickelte Software verwendet, so wird diese im Test erläutert oder darauf verwiesen. Der Quellcode der jeweiligen Software liegt dieser Arbeit bei.
- Die effektiv verwendeten Statistik-Merkmale und signifikanten Resultat-Parameter sind im Paragraph «Fazit» innerhalb Kapitel 3.3.4.1 auf Seite 94 aufgelistet.

Um feststellen zu können, ob sich auf eine längere Zeit ein bestimmtes Verhalten ändert, werden die Tests in verschiedenen Zeitspannen durchgeführt. Dabei gibt es 3 verschiedene Zeitspannen: Ultrakurzzeit-, Kurzzeit und Langzeit-Tests.

Name	Dauer	Dauer t (math.)
Ultrakurzzeit	kleiner als oder genau 60 Sekunden	$t \leq 60s$
Kurzzeit	grösser als 60 Sekunden und kleiner als oder genau 5 Minuten	$60s < t \leq 5min$
Langzeit	grösser als 5 Minuten und kleiner als oder genau 1 Stunde	$5min < t \leq 1h$

Tabelle 3.30: Zeitspannen, in denen die Tests, wenn nötig, durchgeführt werden.

3.3.1.1 Einheiten für Speicher und Netzwerkübertragungen

In diesem Abschnitt werden die für die Speichergrössen und Netzwerkübertragungen verwendeten Einheiten und ihre Relationen zueinander aufgeführt.

Geschwindigkeiten:
Bit

- Bit/s
- kBit/s (1 kBit/s sind 1000 Bit/s)
- MBit/s (1 MBit sind 1000 kBit/s)

Byte (1 Byte sind 8 Bit)

- Byte/s
- kByte/s (1 kByte/s sind 1000 Byte/s)
- MByte/s (1 MByte/s sind 1000 kByte/s)

Mengen:
Bit

- Bit
- kBit (1 kBit sind 1000 Bit)
- MBit (1 MBit sind 1000 kBit)

Byte (1 Byte sind 8 Bit)

- Byte
- kByte (1 kByte sind 1000 Byte)
- MByte (1 MByte sind 1000 kByte)

Bei den Geschwindigkeiten handelt es sich, falls es nicht anders angegeben wird, um Geschwindigkeiten auf Layer 2. Das heisst, dass nur die Bits, die zu einem Layer-2-Frame gehören, zählen und u.a. die Bits der Präambel, SFD und IFG davon ausgeschlossen sind.

3.3.1.2 Generierung von Datenverkehrströmen / Netzwerklast

In diesem Kapitel werden sämtliche Typen von Netzwerklast, die in dieser Arbeit vorkommen, beschrieben und Erzeugung erläutert. Für die Generierung der Netzwerklast werden 2 verschiedene Lasttypen angewandt:

Der Netzwerkverkehr besteht...

- **...nur aus kleinen Frames** (Lasttypen-Bezeichnung: «MIN»)
  Layer-2-Frames / UDP-Pakete: 14 Byte Ethernet Header + 46 Byte Payload (davon bei UDP 8 Byte UDP-Header) + 6 Byte RCT + 4 Byte Frame Check Sequence

= **70 Byte** (wenn ein VLAN-Tag gebraucht wird, fallen 4 Byte mehr an) [54]
TCP-Pakete: 14 Byte Ethernet Header + 53 Byte Payload (davon 20 Byte IP + 32 Byte TCP + 1 Byte Payload (es können über Sockets keine Frames / Pakete ohne Payload versendet werden)) + 6 Byte RCT + 4 Byte Frame Check Sequence = **77 Byte** und mit einem VLAN-Tag 81 Byte. Bezüglich der Grösse von TCP-Paketen gibt es aufgrund der Beschaffenheit von TCP-Sockets Ungenauigkeiten, worauf im Paragraph «TCP-Sockets und bytegenaue Grössen» auf Seite 59 näher eingegangen wird.
Hierbei handelt es sich um einen der Extremfälle, da weitaus weniger Zeit für die Duplikaterkennung vorhanden ist, bis das nächste Frame ankommt. In diesem Fall wird die maximal mögliche Anzahl an Frames / Paketen innerhalb einer Sekunde generiert, weshalb der PRP-1 stack hier am meisten Rechenoperationen ausführen muss, was in einer hohen CPU-Last resultiert.

- **...nur aus grossen Frames** (Lasttypen-Bezeichnung: «MAX»)
  Mit RCT und ohne VLAN-Tag 1518 Byte. In dieser Arbeit wird der RCT mit in die maximale Framegrösse miteinbezogen, die Grösse von 1518 Byte der Standard ist und somit nicht die gesamte Netzwerkperipherie neu parametrisiert werden muss. Des Weiteren ist beim PRP-1 stack eine MTU von 1494 Byte gesetzt, was Frames grösser als 1518 Byte ausschliesst.
  Ein grosses Frame setzt sich somit folgendermassen zusammen: 14 Byte Ethernet-Header + 1494 Byte Payload + 6 Byte RCT + 4 Byte Frame Check Sequence = **1518 Byte**
  Im PRP-1-Standard wird angenommen, dass die jede Netzwerkkomponente eine von ISO/IEC/IEEE 8802-3:2014 vorhergesehene «Oversize»-Grösse von 1528 Byte (max. Frame von 1518 Byte + 4 Byte VLAN-Tag + 6 Byte RCT) unterstützt [54].
  Der zweite der Extremfälle, da permanent grosse Daten verarbeitet werden müssen. Hier wird das Netzwerk am stärksten belastet.

Die Grösse der Frames / Pakete und dessen Header wird in Kapitel 3.2.4.2 auf Seite 53 erläutert und nachgewiesen.

Es werden 3 verschiedene Übermittlungsarten verwendet: Ethernet (reine Layer-2-Frames), TCP und UDP. Für jeden dieser Übermittlungsarten gibt es die 2 vorhin erläuterten Lasttypen.

Dies ergibt total 6 unterschiedliche Netzwerklastarten. Mittels der Applikation shck (siehe Kapitel 3.2.4.2 auf Seite 53) ist es möglich, jeden dieser Datenströme mit einer Genauigkeit zur Einheit Byte generieren zu können.

Des Weiteren ist es möglich mit shck ein Lastmuster zu bestimmen, bei der jedes Frame / Paket eine andere Grösse hat (Lasttypen-Bezeichnung «CUSTOM»). Die Grössen werden aus einer Textdatei, die shck beiliegt und editiert werden kann, Zeile für Zeile in einer Schleife ausgelesen. Wie dies ausgeführt werden kann ist in der «README»-Datei, die shck im selben Ordner beiliegt, beschrieben.

Während der Planung der Testszenarien wurde eine Last mit zufälligen, normalverteilten Grössen in Betracht gezogen. Nach einigen Diskussionen und Statistikanalysen

stellte sich jedoch heraus, dass daraus resultierende Ergebnisse zu generisch und damit für eine konkrete Interpretation nicht genügend aussagekräftig sind.

Es gibt für jeden Anwendungsfall ein anderes Lastmuster, beispielsweise Internet oder Substation Automation, auf welche in dieser Arbeit nicht eingegangen werden. Mit den beigelieferten Applikationen (meas und shck) ist es jedoch möglich, spezifische, selbst erstellte Lastmuster für die Netzwerklastgenerierung mit shck anzuwenden.

Bezeichnung der Datenströme
Die generierten Netzwerklasten erhalten je eine eindeutige Identifikation, welche wie folgt aufgestellt ist:

- Nummer des Szenarios, in dem die Last generiert wird
  - Gehört die Last zu keinem Szenario, anstelle einer Nummer eine passende Folge zweier Buchstaben gewählt.
  - Übermittlungsart in Form eines Buchstabens
    - «E» für Ethernet / Layer-2-Frames
    - «T» für TCP-Pakete
    - «U» für UDP-Pakete
- Lasttyp («MIN», «MAX») (siehe Kapitel 3.3.1.2 auf Seite 74) als dreistelliges Kürzel
  - «MIN» für Frames / Pakete mit minimaler Grösse
  - «MAX» für Frames / Pakete mit maximaler Grösse
- Zeitspanne (siehe Kapitel 3.3.1 auf Seite 73), in der die Netzwerklast generiert wird, in Form eines Buchstabens. Sofern es nicht anders aufgeführt ist, beträgt die Dauer für Ultrakurzzeit-Messungen 60s, für Kurzzeit-Messungen 5min und für Langzeit-Messungen 1h. Wird anstelle einer Zeitspanne eine Anzahl an Frames / Paketen verwendet, so wird hier anstelle des Buchstabens die Anzahl gefolgt von einem «x» aufgeführt.
  - «U» für Ultrakurzzeit
  - «K» für Kurzzeit
  - «L» für Langzeit

Zum Beispiel hat eine Netzwerklast, die in Szenario 01 generiert, via TCP versendet wird, aus Paketen mit maximaler Grösse besteht und während 5min gemessen wird das folgende Kürzel: 01.T.MAX.K

3.3.1.3 Theoretische Grenzwerte

In diesem Abschnitt wird pro Lasttyp berechnet, was die maximal mögliche Netzwerklastung sein würde. Somit kann man anhand der Resultate feststellen, ob beim jeweiligen Testfall ein Engpass beim Prozessor oder bei der Netzwerkverbindung entsteht.

Für Frames / Pakete mit zufälliger Grösse kann dies aufgrund ihrer Natur nicht konkret berechnet werden, jedoch muss dessen maximale Bandbreite zwischen denen der minimalen und maximalen Frames liegen.

Die maximale Übertragungsgeschwindigkeit in der Umgebung dieser Arbeit beträgt auf dem Physical-Layer (Layer 1) 100 MBit/s. Vor jedem Frame / Paket wird die Präambel und der Start of Frame Delimiter (SFD) (zusammen 8 Bytes) angehängt [9]. Nach jedem Frame / Paket wird bei der Datenübertragung ein «Interframe Gap» (IFG) eingefügt, das bei 100 MBit/s 12 Bytes gross ist, da bei Fast Ethernet (100 Mbit/s) die IFG 960ns dauern muss [58]:

$$IFG = \frac{100'000'000\,Bit/s}{1'000'000'000\,ns} * 960\,ns = 96\,Bit = 12\,Bytes$$

Die Bitraten der folgenden Berechnungen beziehen sich auf den Data-Link-Layer (Layer 2).

Frames mit minimaler Grösse
Ein Frame mit minimaler Grösse nimmt bei der Übertragung effektiv 84 Byte ein (7 Byte Präambel + 1 Byte SFD + 64 Byte Ethernet Frame + 12 Byte IFG), was 672 Bit beträgt.

$$Anzahl\,Frames/s = \frac{100'000'000\,Bit/s}{672\,Bit} = 148'809.52381\,Frames/s \approx 148'810\,Frames/s$$

Multipliziert man diese Anzahl an Frames lediglich mit der Grösse eines minimalen Ethernet Frames (64 Byte = 512 Bit), erhält man die effektive Bitrate:

$$148'810\,Frames/s * 368\,Bit = 76'190'720\,Bit/s \approx 76.2\,MBit/s$$

Somit liegt die maximale Bandbreite bei einer Nutzlast, die lediglich aus minimal grossen Frames / Paketen besteht bei ca. 76.2 MBit/s.

Mit dem selben Rechnungsweg ergibt die maximale Bandbreite mit minimal grossen TCP-Paketen (beträgt bei der Übertragung effektiv 97 Bytes) 79.4 MBit/s.

Frames mit maximaler Grösse
Ein Frame mit maximaler Grösse nimmt effektiv 1538 Byte ein (7 Byte Präambel + 1 Byte SFD + 1518 Byte Ethernet Frame + 12 Byte IFG), was 12'304 Bit beträgt.

$$Anzahl\,Frames/s = \frac{100'000'000\,Bit/s}{12'304\,Bit} = 8'127.43823147\,Frames/s \approx 8128\,Frames/s$$

Multipliziert man diese Anzahl an Frames lediglich mit der Grösse eines maximalen Ethernet Frames (1518 Byte = 12'144 Bit), erhält man die effektive Bitrate:

$$8128\,Frames/s * 12'144\,Bit = 98,706,432\,Bit/s \approx 98.7\,MBit/s$$

Somit liegt die maximale Bandbreite bei einer Nutzlast, die lediglich aus maximal grossen Frames / Paketen besteht bei ca. 98.7 MBit/s.

3.3.1.4 Einschwingzeit / Steady State

Es folgt eine Analyse für jede Übertragungsart (Ethernet, TCP, UDP), bei der die Dauer bis sich das System im eingeschwungenen Zustand befindet ermittelt wird. Des Weiteren wird pro Übertragungs- und Lastart determiniert welche Grösse die Schwankungen im eingeschwungenen Zustand betragen. Dieser eingeschwungene Zustand wird auch Steady State genannt.

Dazu wird für jedes Protokoll und jeden Lasttyp eine Netzwerklast mit 100'000 Frames / Paketen mit shck generiert, versendet und mit meas gemessen (meas wird gestartet, dann shck). So wird jeweils der Verbindungsaufbau, Versand und Verbindungsabbau aufgezeichnet. Während der Aufzeichnung werden alle 100ms die aktuellen Performance-Werte gespeichert und am Ende ausgewertet, womit auf eine Zehntel Sekunde genau festgelegt werden kann, wie lange die Einschwingzeit, der Steady State und der Verbindungsabbau dauern.

In den folgenden Grafiken werden die CPU-Last des PRP-1 stacks und die TX-Bitrate des PRP-Netzwerk-Interfaces aufgezeigt.

E.MIN.100000x

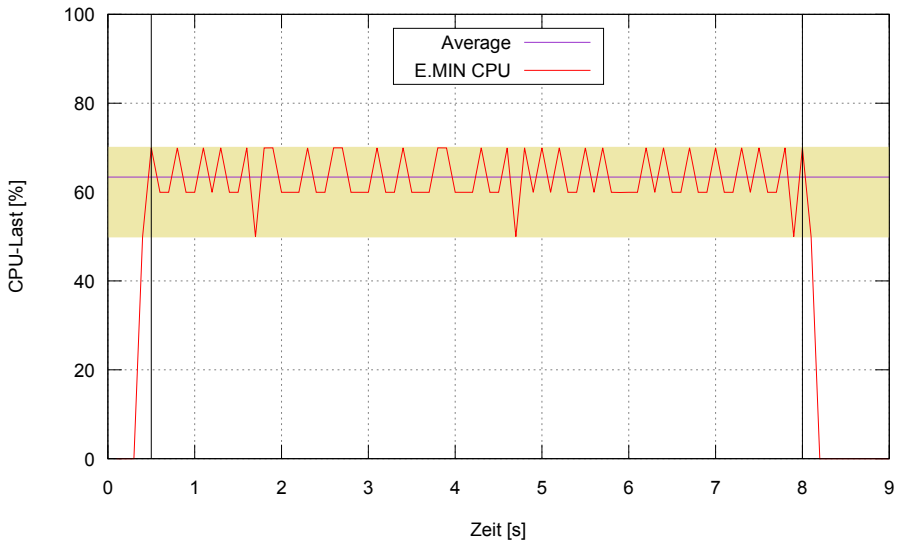


Abbildung 3.19: E.MIN.100000x: Verlauf der CPU-Last des PRP-1 stacks

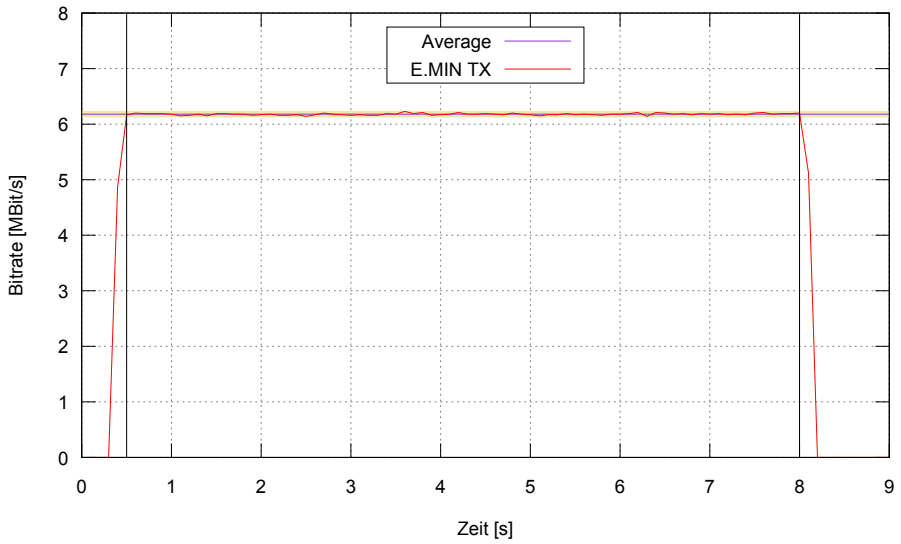


Abbildung 3.20: E.MIN.100000x: Verlauf der TX-Bitrate des PRP-Netzwerk-Interfaces

Bezeichnung	**Wert**
Steady State Dauer von [s] bis [s]	0.5 - 8
Durchschnitt im Steady State CPU [%]	63.37
Schwankungsband im Steady State CPU [%]	20.48
Durchschnitt im Steady State TX [MBit/s]	06.18
Schwankungsband im Steady State TX [MBit/s]	00.13

Tabelle 3.31: E.MIN.100000x: Einschwingzeit / Steady State / Schwankungen

E.MAX.100000x

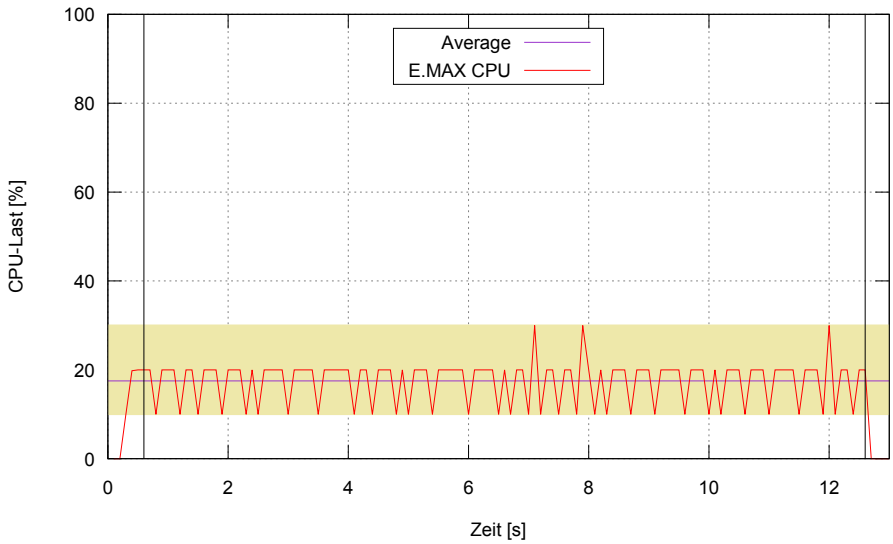


Abbildung 3.21: E.MAX.100000x: Verlauf der CPU-Last des PRP-1 stacks

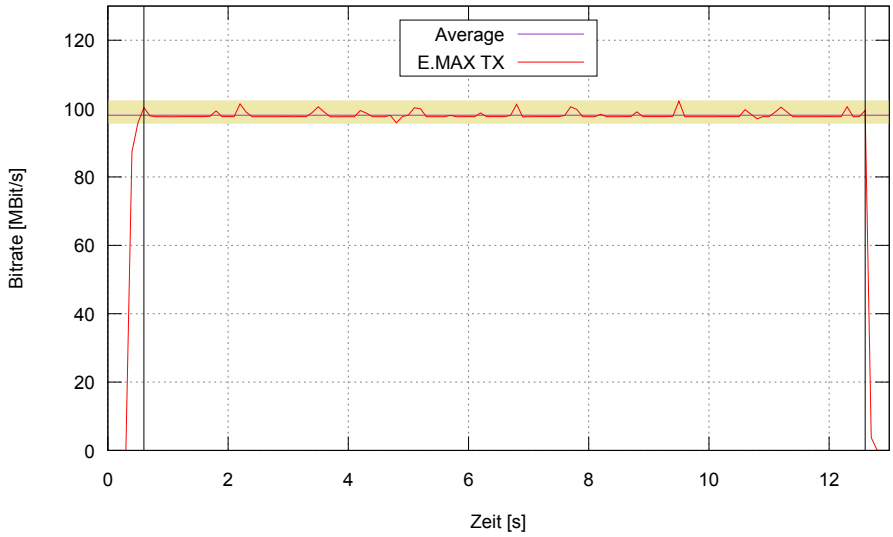


Abbildung 3.22: E.MAX.100000x: Verlauf der TX-Bitrate des PRP-Netzwerk-Interfaces

Bezeichnung	**Wert**
Steady State Dauer von [s] bis [s]	0.6 - 12.6
Durchschnitt im Steady State CPU [%]	17.51
Schwankungsband im Steady State CPU [%]	19.05
Durchschnitt im Steady State TX [MBit/s]	98.07
Schwankungsband im Steady State TX [MBit/s]	06.29

Tabelle 3.32: E.MAX.100000x: Einschwingzeit / Steady State / Schwankungen

T.MIN.100000x

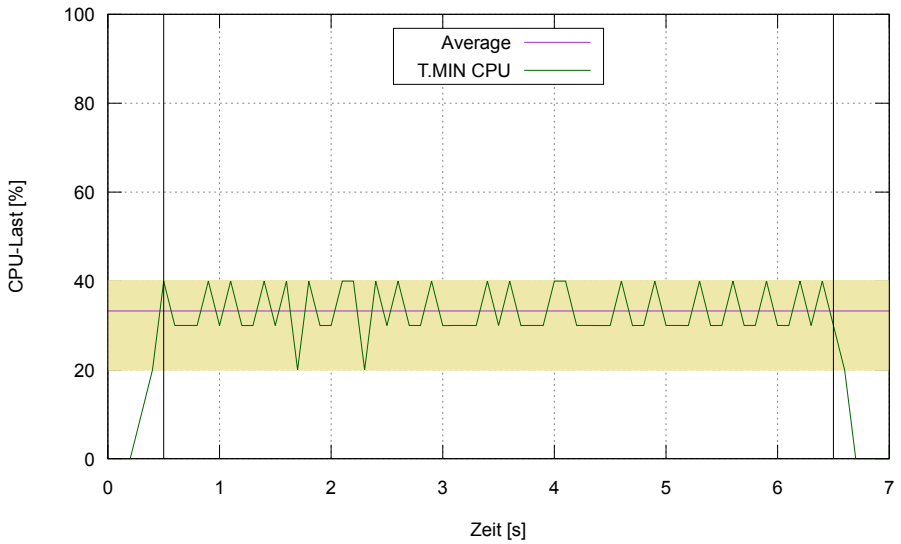


Abbildung 3.23: T.MIN.100000x: Verlauf der CPU-Last des PRP-1 stacks

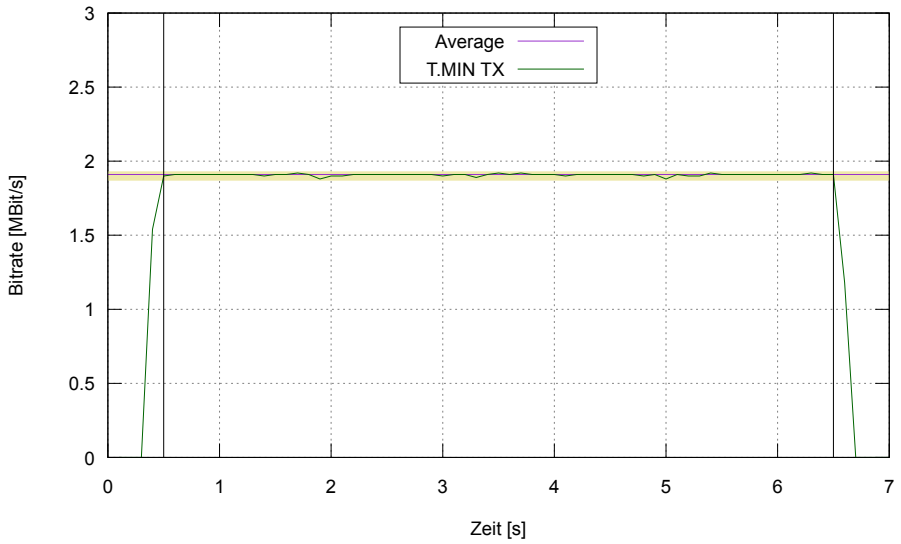


Abbildung 3.24: T.MIN.100000x: Verlauf der TX-Bitrate des PRP-Netzwerk-Interfaces

Bezeichnung	**Wert**
Steady State Dauer von [s] bis [s]	0.5 - 6.5
Durchschnitt im Steady State CPU [%]	33.25
Schwankungsband im Steady State CPU [%]	19.62
Durchschnitt im Steady State TX [MBit/s]	01.91
Schwankungsband im Steady State TX [MBit/s]	00.04

Tabelle 3.33: T.MIN.100000x: Einschwingzeit / Steady State / Schwankungen

T.MAX.100000x

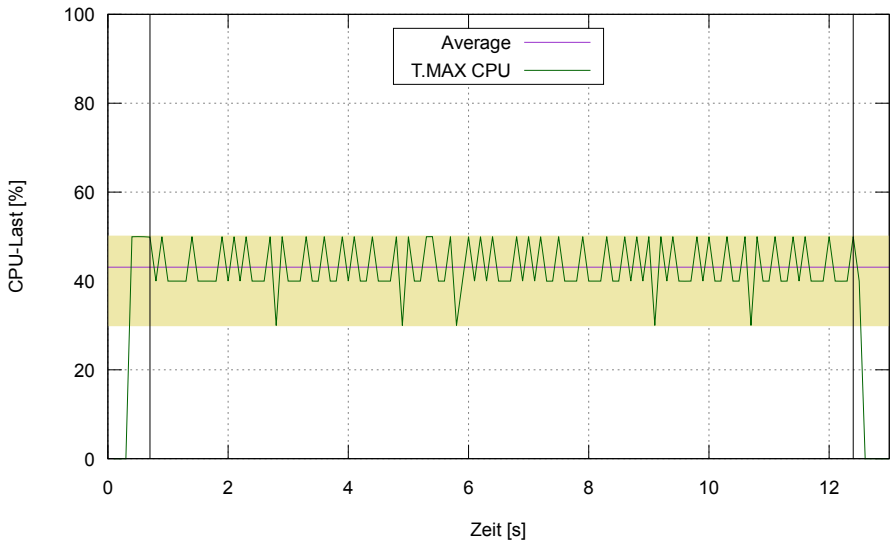


Abbildung 3.25: T.MAX.100000x: Verlauf der CPU-Last des PRP-1 stacks

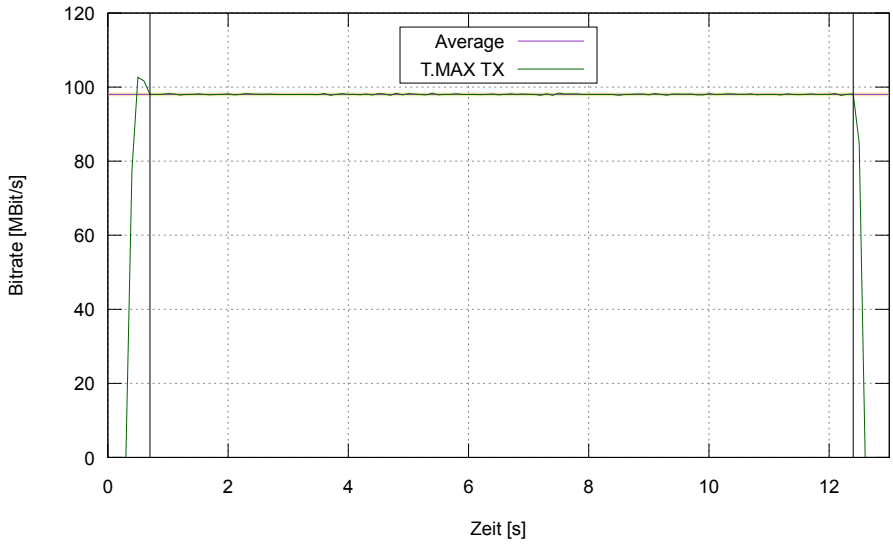


Abbildung 3.26: T.MAX.100000x: Verlauf der TX-Bitrate des PRP-Netzwerk-Interfaces

Bezeichnung	**Wert**
Steady State Dauer von [s] bis [s]	0.7 - 12.4
Durchschnitt im Steady State CPU [%]	43.10
Schwankungsband im Steady State CPU [%]	20.00
Durchschnitt im Steady State TX [MBit/s]	98.05
Schwankungsband im Steady State TX [MBit/s]	01.14

Tabelle 3.34: T.MAX.100000x: Einschwingzeit / Steady State / Schwankungen

U.MIN.100000x

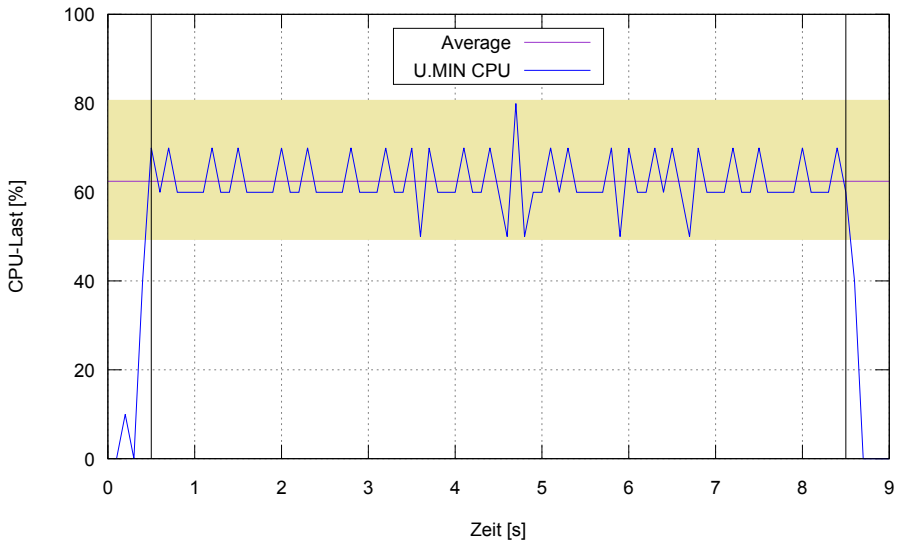


Abbildung 3.27: U.MIN.100000x: Verlauf der CPU-Last des PRP-1 stacks

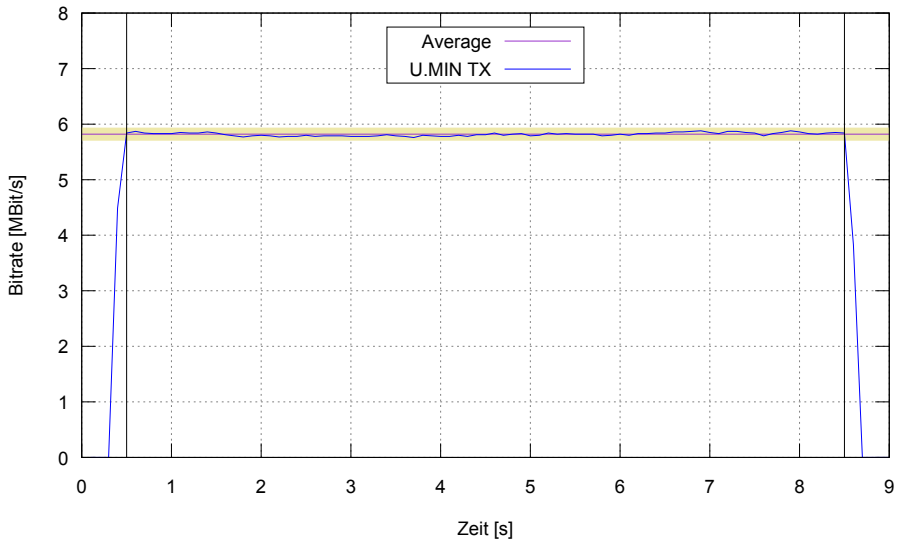


Abbildung 3.28: U.MIN.100000x: Verlauf der TX-Bitrate des PRP-Netzwerk-Interfaces

Bezeichnung	**Wert**
Steady State Dauer von [s] bis [s]	0.5 - 8.5
Durchschnitt im Steady State CPU [%]	62.42
Schwankungsband im Steady State CPU [%]	30.00
Durchschnitt im Steady State TX [MBit/s]	05.82
Schwankungsband im Steady State TX [MBit/s]	00.22

Tabelle 3.35: U.MIN.100000x: Einschwingzeit / Steady State / Schwankungen

U.MAX.100000x

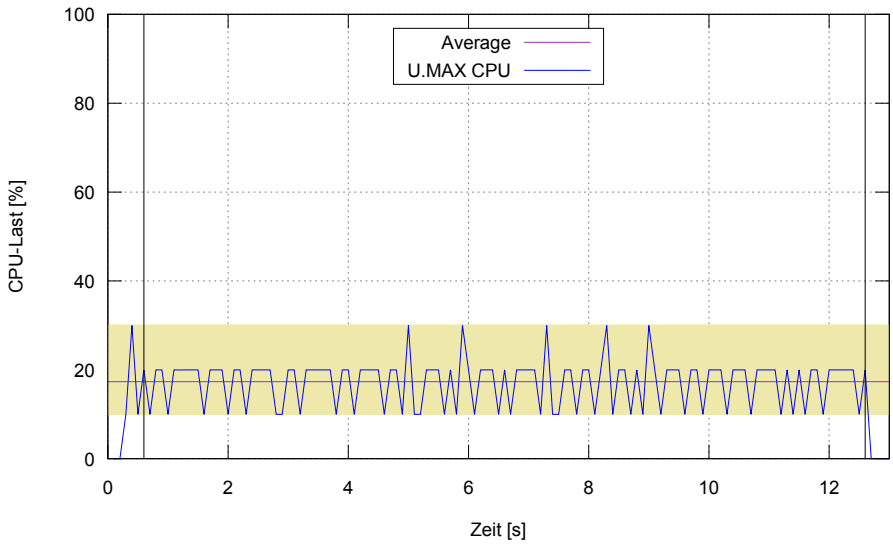


Abbildung 3.29: U.MAX.100000x: Verlauf der CPU-Last des PRP-1 stacks

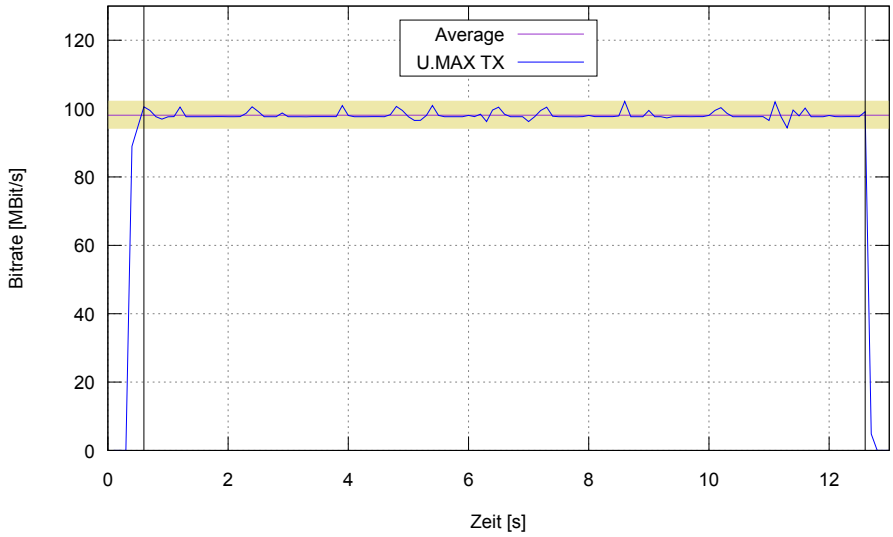


Abbildung 3.30: U.MAX.100000x: Verlauf der TX-Bitrate des PRP-Netzwerk-Interfaces

Bezeichnung	**Wert**
Steady State Dauer von [s] bis [s]	0.6 - 12.6
Durchschnitt im Steady State CPU [%]	17.34
Schwankungsband im Steady State CPU [%]	19.81
Durchschnitt im Steady State TX [MBit/s]	98.06
Schwankungsband im Steady State TX [MBit/s]	07.08

Tabelle 3.36: U.MAX.100000x: Einschwingzeit / Steady State / Schwankungen

Fazit
Die längste Einschwingzeit über alle Lasttypen beträgt 0.7s. Das bedeutet, dass wenn meas und shck direkt nacheinander gestartet werden, meas garantiert nach 0.7s einen eingeschwungenen Zustand misst.

Für die Messungen bedeutet das, dass etwa 1 Sekunde (0.7s aufgerundet) nach dem Start der Netzwerklastgenerierung mit der Messung begonnen werden kann, um garantiert nur Werte aus dem Steady State auszulesen. Um dies über alle Messwerte zu garantieren, wird die Messung vor der Netzwerklastgenerierung beendet.

In den zuvor aufgeführten Graphen ist u.a. sichtbar, dass die TX-Übertragungsrate manchmal über dem theoretischen Grenzwert (siehe Kapitel 3.3.1.3 auf Seite 77) liegt. Dies hängt mit der gewählten Intervalldauer zusammen. Auf diese Tatsache wird im nächsten Kapitel auf dieser Seite eingegangen.

3.3.2 Einfluss von gewählten Intervallgrössen

Die Messgenauigkeit bezüglich der Netzwerkstatistiken wird durch einige unumgänglichen Faktoren beeinträchtigt:

- Verzögerung durch Mechanismen des Betriebssystems
  - Context-Switches
  - Scheduler bevorzugt höher priorisierte Prozesse. Dies aufgrund des Umstandes, dass die meas-Applikation grösstenteils im Userspace betrieben wird. Prozesse, die grössere Zeitabschnitte im Kernelspace betreiben, werden bevorzugt.
- Verzögerungszeit, die während des Auslesens der Werte aus dem /proc-Dateisystem entsteht. Detailliert wird darauf in Paragraph «Minderung von Messfehlern / Optimierungen» auf Seite 46 näher eingegangen.

Um aufzuzeigen, dass Messungen zunehmend ungenau werden, je kürzer ein Messintervall gewählt wird, zeigt die nachfolgende Tabelle.

Intervallgrösse	Netzwerk-interface	Minimum	Durchschnitt	Maximum
0.01s	prp1	56.08	98.12	114.91
	eth0	95.21	98.71	104.84
	eth1	95.46	98.71	104.53
0.1s	prp1	95.11	98.06	101.99
	eth0	98.51	98.71	99.27
	eth1	98.04	97.71	99.27
1s	prp1	97.68	98.05	98.47
	eth0	98.66	98.71	98.72
	eth1	98.65	98.71	98.72
10s	prp1	98.01	98.06	98.10
	eth0	98.70	98.71	98.71
	eth1	98.70	98.71	98.71

Tabelle 3.37: TX-Bitrate pro Intervallgrösse und Netzwerk-Interface [MBit/s] shck -s MAX -t UDP

Die gezeigten Messwerte entstanden bei einer maximalen Netzwerklast (Senden von grösstmöglichen UDP-Paketen). Bei 4 verschiedenen Intervallgrössen wurden je 200 Messintervalle aufgezeichnet.

Betrachtet man die Maxima, wird ersichtlich, dass je kürzer ein Messintervall gewählt wird, diese von theoretischen Grenzwerten (siehe Kapitel 3.3.1.3 auf Seite 77) abweichen, respektive diese signifikant überschreiten.

Mit der nachfolgenden Grafik wird aufgezeigt, dass der Messfeher (roter Abschnitt) immer die selbe Länge aufweist, unabhängig von der Länge des Messintervalls.

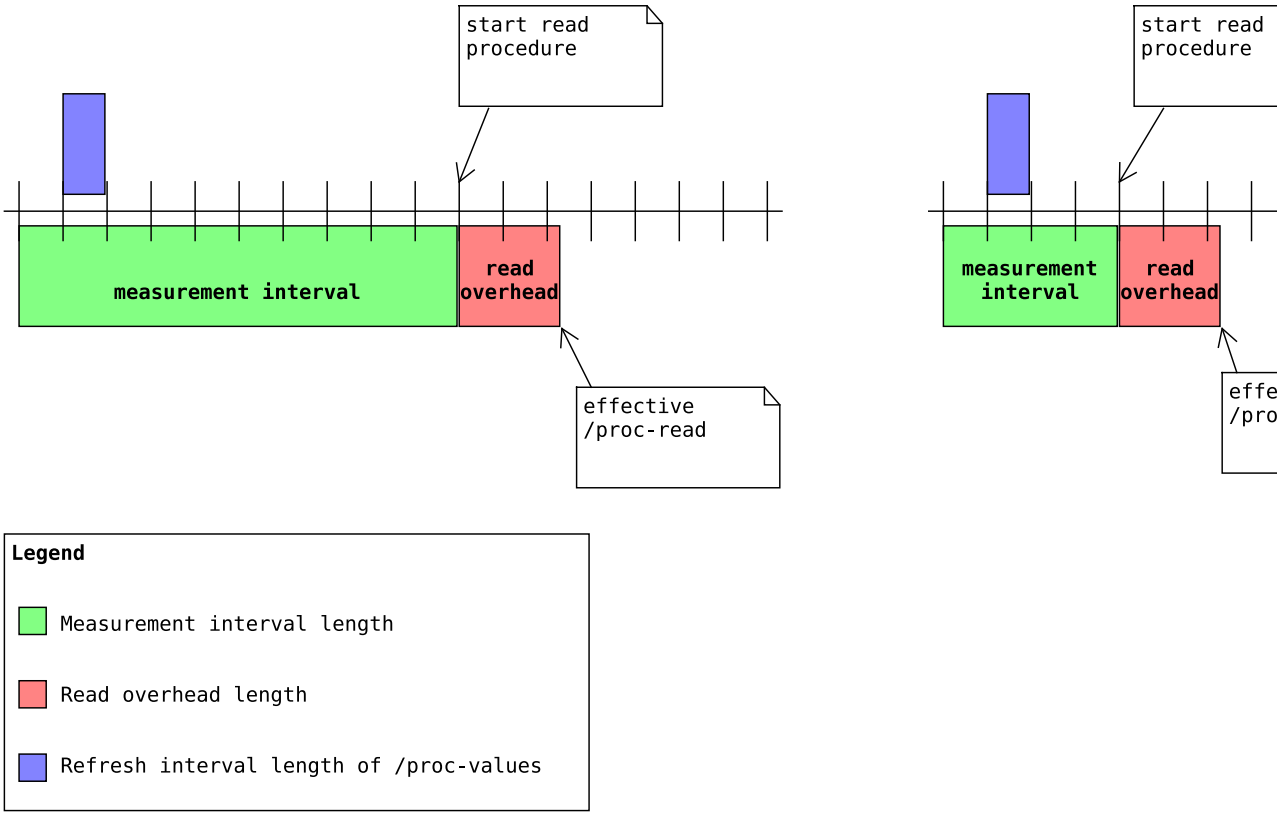


Abbildung 3.31: U.MAX.100000x: Verlauf der TX-Bitrate des PRP-Netzwerk-Interfaces

3.3.3 Ablauf eines Szenarios zur Performanceermittlung

Die Messungen werden in Form von Bash, Python-Scripts und C-Applikationen vorbereitet (Quellcode siehe Kapitel 14 auf Seite 165), um automatisiert ausgeführt werden zu können.

Der Vorgang einer Messung lautet wie folgt:

1. Starten der Netzwerkgenerierung auf einem Rechner, sodass sie etwas länger als die gewählte Zeitspanne (Ultrakurz-, Kurz- oder Langzeit) Netzwerkverkehr stattfindet. Generiert werden die Datenströme mit der Applikation shck. Da so die Netzwerklast immer mit der selben Applikation generiert wird, kann jeweils nur ein Faktor bei der Erzeugung verändert werden, was die Vergleiche der Resultate aussagekräftiger macht. Würde man shck und iperf3 für unterschiedliche Szenarien verwenden und dann diese Szenarien vergleichen, könnte es sein, dass die festgestellten Unterschiede von den Differenzen zwischen shck und iperf3 beeinflusst werden, was das Isolieren eines einzelnen Faktors verhindern würde.
   Beispiel: Vergleich bezüglich Paketgrösse: Generierung von Last mit minimaler Paketgrösse via TCP mit shck und Generierung von Last mit maximaler Paketgrösse via TCP mit iperf3 anstelle mit shck. Würde man hier auch die Last mit maximaler

Paketgrösse mit shck erzeugen, so würde man garantiert nur einen Faktor (Paketgrösse) bei der Netzwerklast verändern.

2. Kurz darauf (ca. 5 Sekunden) werden gleichzeitig die Messungen auf allen beteiligten Servern (meistens 1 Client und 1 Server) gestartet. Zum Startzeitpunkt der Messungen befindet man so sich sicher bereits an einem Punkt, an dem sich ein konstanter Zustand (Steady State) eingependelt hat. Gemessen wird die CPU-Last, welche der PRP-1 stack generiert und die Netzwerkbelastung. Verwendet wird für das Aufzeichnen der CPU- und Netzwerk-Daten die Applikation meas.

3. Während der gewählten Zeitspanne wird die CPU-Last und die Netzwerkbelastung gemessen. Diese Daten werden für eine spätere Analyse ausgelagert.
   Die Messungen werden so durchgeführt, dass innerhalb der Zeitspanne immer nach einer bestimmten Periode die Durchschnitts-Werte dieser Periode gespeichert werden. So lässt sich schlussendlich sagen, wann Minimum und Maximum eingetreten sind und wie viel der gesamte Durchschnitt beträgt. Es wird eine Intervallgrösse von 1 Sekunde gewählt, womit man schlussendlich Messdaten für die komplette Zeitspanne und für jede Sekunde innerhalb der Zeitspanne hat. Anhand dieser Intervallgrösse wird zudem eine aussagekräftige Genauigkeit der Messwerte erzielt (siehe Kapitel 3.3.2 auf Seite 85).

4. Ist die Zeitspanne abgelaufen, werden die Messungen beendet. Wichtig ist hier, dass erst nach dem Abschluss der Messung mit der Generierung von Netzwerkverkehr aufgehört wird, da ansonsten die Messung nicht während einem Steady State statt findet.

Sofern es nicht anders erwähnt wird, erfolgt die Netzwerkgenerierung (shck-Client) auf dem Server «srv01» während auf «srv02» der shck-Server läuft. Gemessen wird dabei auf beiden Hosts. Gestartet wird das Ganze anhand eines Bash-Scripts auf «srv03», welches über das Netzwerk Ext via SSH die Messungen auf beiden Hosts beinahe zeitgleich (direkt nacheinander) startet.

Die Werte werden jeweils wie folgt berechnet:

- **CPU-Last**
  Innerhalb einer Zeitspanne wird aufgezeichnet, wie lange ein Prozess den Prozessor beansprucht hat. Diese Zeit wird durch die Dauer der Zeitspanne dividiert, was die CPU-Last in Prozent, die der Prozess beansprucht hat, ergibt. Hierbei handelt es sich lediglich um Single-Threaded-Prozesse, da der PRP-1 stack und die Tools zur Generierung der Netzwerklast single-threaded sind. meas behandelt aus diesem und aus Zeitgründen keine Multi-Threaded-Applikationen.

- **Arbeitsspeicher-Nutzung**
  Der PRP-1 stack verwendet stets die gleiche Menge an Arbeitsspeicher, nämlich: VSZ 7528 kByte und RSS 3188 kByte. Diese Grösse wurde beim Test mit jeder in dieser Arbeit verwendeten Netzwerklast und beim Beobachten über Monate festgestellt. Aus diesem Grund wird in dieser Arbeit nicht weiter auf die

Arbeitsspeicher-Nutzung eingegangen. Es kann sein, dass diese Grössen auf verschiedenen Hosts variieren, jedoch verändern sich die Werte bezüglich der Arbeitsspeichernutzung nach dem Start des PRP-1 stacks nicht mehr.

- **Netzwerkbelastung**
  Zu Beginn und Ende der Messung wird jeweils ausgelesen, wie viele Bytes empfangen und versendet wurden. Diese Differenz wird dann durch die Messdauer in Sekunden dividiert, um danach die Down- und Uploadrate in Bytes pro Sekunde zu erhalten.

Konkret werden folgende Werte je ein mal pro zu untersuchenden Prozess (zum Beispiel PRP-1 stack) pro Messung in einem Analyse-Ergebnis aufgeführt:

Wert	Bedeutung	Einheit
CPU-Last Minimum	Wert der Periode innerhalb der Zeitspanne, in der das untersuchte Objekt am wenigsten Zeit vom Prozessor beansprucht hat.	%
Systemtime Minimum		s
Usertime Minimum		s
CPU-Last Maximum	Wert der Periode innerhalb der Zeitspanne, in der das untersuchte Objekt am meisten Zeit vom Prozessor beansprucht hat.	%
Systemtime Maximum		s
Usertime Maximum		s
CPU-Last Durchschnitt	Durchschnitt der Werte alle Perioden. Besagt wie viel der Zeit innerhalb der Zeitspanne die CPU für das untersuchte Objekt benötigt hat.	%
Systemtime Durchschnitt		s
Usertime Durchschnitt		s

Tabelle 3.38: Beschreibung der Werte vom Ergebnis einer Messung: CPU

Da bei der Konfiguration der Endsysteme und der Netzwerkumgebung sichergestellt wurde, dass nur gewollter Netzwerkverkehr besteht, kann man die Netzwerkbelastung pro Netzwerkinterface anstelle für jeden Prozess einzeln festhalten.

Wert	Bedeutung	Einheit
RX-Bitrate (Download) Minimum	Kleinste Netzwerkbelastung der Periode innerhalb der Zeitspanne, in der das Netzwerkinterface am wenigsten das Netzwerk belastet hat.	MBit/s
TX-Bitrate (Upload) Minimum		
RX-Bitrate (Download) Maximum	Grösste Netzwerkbelastung der Periode innerhalb der Zeitspanne, in der das Netzwerkinterface am meisten das Netzwerk belastet hat.	MBit/s
TX-Bitrate (Upload) Maximum		
RX-Bitrate (Download) Durchschnitt	Durchschnittliche Netzwerkbelastung über die ganze Zeitspanne, in der gemessen wurde.	MBit/s
TX-Bitrate (Upload) Durchschnitt		

Tabelle 3.39: Beschreibung der Werte vom Ergebnis einer Messung: Netzwerkbelastung pro Netzwerkinterface

3.3.4 Szenarien

Es folgt eine kurze Aufführung der wichtigsten Eckdaten zum Ursprungszustand der Testumgebung, welche detaillierter im Kapitel 3.1 auf Seite 23 aufgeführt ist:

- 3x Server «HP ProLiant DL140» mit Debian, je an beiden Netzwerken des PRP-Netzes angeschlossen.
- Jede Netzwerkkarte der Server, die am PRP-Netz angeschlossen ist, wird mit 100 MBit/s betrieben. Des Weiteren sind in diesen NICs jegliche Offload-Mechanismen (siehe Kapitel 2.2 auf Seite 19) deaktiviert.
  - Bei den IP-Adressen handelt es sich um manuell zugewiesene Adressen.
- Sämtliche Applikationen, Ordnerstrukturen und Skripts sind auf allen 3 Servern in der gleichen Ausführung vorhanden.
- Innerhalb des PRP-Netzes findet kein Netzwerkverkehr statt, sofern auf den Servern keine Netzwerk-Applikationen ausgeführt werden.
- Neben den 3 aufgeführten Servern und den 2 Switches befinden sich keine weiteren Netzwerkkomponenten im PRP-Netz, ausser es wird im entsprechenden Szenario erwähnt, dass der Umgebungsaufbau gegenüber dem Standard variiert.

Das Ergebnis eines Szenarios ist im Kapitel 4 auf Seite 104 im entsprechenden Unterkapitel aufgeführt. Ausnahme bildet die Erörterung signifikanter Resultat-Parameter im nächsten Kapitel, weil das Ergebnis dieses Vergleichs als Grundlage für den Aufbau der Szenarien dient.

Der Quellcode aller erstellten Bash-, Python-Skripte und C-Applikationen sämtlicher Szenarien ist im Kapitel 14.3 auf Seite 165 aufgelistet.

3.3.4.1 Erörterung signifikanter Resultat-Parameter: UDP / Ethernet Vergleich & Notwendigkeit der Langzeit-Tests

Aufgrund der ähnlichen Verhaltensweisen von UDP-Segmenten und Layer-2-Frames besteht Grund zur Annahme, dass diese sich auch ähnlich auf die Performance auswirken. Daher wird in diesem Abschnitt untersucht, ob und was für eine Differenz bei der Messung von UDP- und Ethernet-Datenverkehr mit PRP entsteht. Aus dieser Untersuchung soll sich heraus stellen, ob es nötig ist, den Layer-2- vom UDP-Verkehr getrennt zu begutachten oder ob es genügt den UDP-Verkehr zu analysieren und daraus Ergebnisse für den Layer-2-Datenverkehr abzuleiten. Sollte es nach der Analyse in diesem Abschnitt nicht von Nöten sein, den Layer-2-Datenstrom separat zu begutachten, kann für die darauf folgenden Untersuchungen lediglich TCP und UDP für die Generierung der Netzwerklast verwendet werden.

Des Weiteren wird ermittelt, ob Langzeit-Tests (Dauer jeweils 1h) nötig sind oder ob sie die selben Resultate wie Kurzzeit-Tests (Dauer je 5min) erbringen. Würde es sich herausstellen, dass die Langzeit-Tests nicht von Nöten sind, so würde dies eine grosse Zeiteinsparung mit sich ziehen.

Der Vergleich findet im PRP-Netzwerk statt, weil diese Umgebung im Fokus dieser Arbeit liegt. Die Konfiguration der Umgebung kann in Kapitel 3.1 auf Seite 23 eingesehen werden. Es ist davon auszugehen, dass die Unterschiede zwischen Ethernet- und UDP-Traffic in einem Nicht-PRP-Umfeld identisch ausfallen, weil die Umgebung die beiden Typen äquivalent beeinflusst.

Betrachtet wird die Belastung der CPU durch den PRP-1 stack und der Netzwerkinterfaces. Davon werden jeweils die kleinsten und grössten sowie die durchschnittlichen (in der folgenden Tabelle als «Avg» betitelt) Werte, die innerhalb einer Periode von 1s während der Zeitspanne erzielt wurden, aufgelistet. Dabei werden die Netzwerklasten nebeneinander aufgeführt, welche sich in der Übermittlungsart unterscheiden. Auf die Arbeitsspeicher bezogenen Werte wird hier nicht eingegangen, da der PRP-1 stack keine dynamische Speicherallozierung verwendet (siehe Kapitel 2.1.1 auf Seite 17).

In den folgenden Tabellen werden die generierten Netzwerklasten mit einer eindeutigen Bezeichnung identifiziert. Wie diese Bezeichnung aufgebaut ist, ist in Kapitel 3.3.1.2 auf Seite 76 aufgeführt.

CPU-Belastung

Zu beachten ist, dass in der folgenden Tabelle die Ergebnisse einer Periode von 1s abstammen. Dies bedeutet, dass z.B. die kleinste CPU-Last nur während 1s innerhalb der kompletten Zeitspanne von 5min vorgekommen sein könnte. Der durchschnittliche Wert wird hingegen aus den Ergebnissen aller Perioden innerhalb der Zeitspanne berechnet.

Bezeichnung	CPU-Last [%]			Systemtime [s]			Usertime [s]		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
ER.E.MIN.U	64.00	65.28	66.00	0.53	0.57	0.62	0.05	0.09	0.13
ER.U.MIN.U	62.98	63.96	65.00	0.52	0.56	0.61	0.03	0.08	0.12
ER.E.MIN.K	59.99	65.46	66.00	0.47	0.55	0.61	0.03	0.08	0.14
ER.U.MIN.K	55.00	64.14	66.00	0.47	0.56	0.61	0.04	0.08	0.12
ER.E.MIN.L	58.00	62.35	66.00	0.46	0.55	0.62	0.02	0.07	0.14
ER.U.MIN.L	57.00	61.55	66.00	0.45	0.54	0.62	0.01	0.07	0.13
ER.E.MAX.U	16.00	17.52	18.00	0.14	0.16	0.18	0.00	0.01	0.04
ER.U.MAX.U	16.00	17.35	19.00	0.13	0.15	0.18	0.00	0.02	0.04
ER.E.MAX.K	16.00	17.71	19.00	0.13	0.16	0.18	0.00	0.02	0.04
ER.U.MAX.K	16.00	17.28	19.00	0.13	0.16	0.18	0.00	0.02	0.05
ER.E.MAX.L	16.00	17.49	19.00	0.12	0.16	0.18	0.00	0.02	0.06
ER.U.MAX.L	16.00	17.26	19.00	0.11	0.16	0.18	0.00	0.02	0.06

Tabelle 3.40: UDP / Ethernet Vergleich: CPU-Last, System- / Usertime

Netzwerk-Belastung

Wie auch bei der CPU-Belastung stammen in der folgenden Tabelle die Ergebnisse einer Periode von 1s ab. Aus Platzgründen werden hier lediglich die durschnittlichen Werte betrachtet.

Da der Client bei der Übertragung von reinen Layer-2-Frames und UDP-Paketen keine Antwort vom Server empfangen und ausser shck nur noch die PRP_Supervision-Frames (siehe Kapitel 2.1 auf Seite 14) das Netzwerk belasten, ist die RX-Rate bei den Interfaces «eth0» und «eth1» verschwindend klein. Bei keiner der unten aufgezeichneten Netzwerklast betrug die RX-Bitrate mehr als 1kBit/s. Beim Interface «prp1» werden PRP_Supervision-Frames nicht gezählt, was bei jeder der unten aufgeführten Netzwerklast für dieses Interface eine RX-Rate von 0 Bit/s ergibt.

Bezeichnung	TX-Bitrate [MBit/s]								
	prp1			eth0			eth1		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
ER.E.MIN.U	8.03	8.08	8.12	9.37	9.43	9.48	9.37	9.43	9.48
ER.U.MIN.U	7.61	7.70	7.77	8.88	8.98	9.06	8.88	8.98	9.06
ER.E.MIN.K	7.43	7.63	7.89	8.67	8.91	9.20	8.67	8.91	9.20
ER.U.MIN.K	6.46	7.36	7.46	7.54	8.59	8.70	7.54	8.59	8.70
ER.E.MIN.L	7.31	7.64	7.89	8.52	8.91	9.20	8.52	8.91	9.20
ER.U.MIN.L	7.07	7.52	7.78	8.25	8.77	9.08	8.25	8.77	9.08
ER.E.MAX.U	97.71	98.06	98.46	98.66	98.71	98.72	98.66	98.71	98.72
ER.U.MAX.U	97.67	98.06	98.47	98.66	98.71	98.72	98.66	98.71	98.72
ER.E.MAX.K	97.64	98.06	98.43	98.66	98.71	98.74	98.66	98.71	98.73
ER.U.MAX.K	97.64	98.06	98.47	98.66	98.71	98.74	98.66	98.71	98.74
ER.E.MAX.L	97.64	98.06	98.51	98.65	98.71	98.77	98.65	98.71	98.76
ER.U.MAX.L	97.63	98.06	98.47	98.65	98.71	98.77	98.64	98.71	98.76

Tabelle 3.41: UDP / Ethernet Vergleich: RX- / TX-Bitrate pro Netzwerkinterface

Interpretation: Auswahl der signifikanten Resultat-Parameter
Es stellt sich heraus, dass die gemessenen Werte innerhalb den selben Lasttypen sehr ähnlich ausfallen.

Lasttyp «MIN» Bezüglich der CPU-Belastung sticht heraus, dass sich Ethernet- und UDP-Netzwerklasten innerhalb des Lasttyps «MIN» am stärksten, nämlich um ein paar Prozent pro Zeitspanne unterscheiden. Dabei ist stets die CPU-Belastung der Ethernet-Netzwerklasten höher als die durch UDP. Die erzielten Bitraten der Netzwerkströme dieses Lasttyps sind weit unter dem entsprechenden theoretischen Grenzwert (siehe Kapitel 3.3.1.3 auf Seite 77), was daraus schliessen lässt, dass hier die CPU der limitierende Faktor ist und somit bei der Lastgenerierung dieses Typs zusammen mit dem PRP-1 stack die komplette CPU-Last beansprucht wird.

Dadurch, dass beim Versand über UDP-Sockets das Betriebssystem häufiger zum Einsatz kommt (Generierung UDP- & IP-Header pro Paket) als beim Versand über Ethernet-Sockets, bleibt weniger CPU-Zeit für den PRP-1 stack übrig, weshalb dieser weniger UDP-Pakete als Ethernet-Frames abarbeiten kann. Daraus resultieren auch die in diesem Lasttyp kleineren Bitraten bei UDP- gegenüber Ethernet-Netzwerklasten.

Lasttyp «MAX» Innerhalb dieses Lasttyps sind kaum Unterschiede zwischen den Übermittlungsarten festzustellen, was sich auch über die Zeitspannen des selben Lasttyps weiter zieht. Die erreichten Bitraten sind sehr nahe am theoretischen Grenzwert (siehe Kapitel 3.3.1.3 auf Seite 77), weshalb bei der Erzeugung von Netzwerklasten dieses Lasttyps das Netzwerk anstelle der CPU der Engpass darzustellen scheint.

Zeitspannen Innerhalb einer Übertragungsart und eines Lasttyps ist über die drei Zeitspannen keine signifikante Änderung festzustellen (was u.a. auch an den schmalen Schwankungsbändern in Kapitel 3.3.1.4 auf Seite 78 zu sehen ist). Da die Ultrakurz-Zeitspanne mit 60s z.B. für die Analyse von Laufzeitunterscheidungen zwischen Netzwerk A und B knapp ausfallen könnte, 5min (Kurzzeit-Zeitspanne) dafür aber aussagekräftigere Resultate liefern würde, besteht Grund dazu, lediglich diese beiden Zeitspannen zu berücksichtigen.

Da sich Übertragungen via TCP von denen per UDP lediglich im Socket und verwendeten Layer-4-Protokoll unterscheiden, wird angenommen, dass für die Performance-Ermittlung mit TCP ebenfalls keine Langzeit-Szenarien nötig sind.

Fazit Aufgrund den vorhin ermittelten Werten genügt es den UDP-Verkehr zu analysieren und daraus Ergebnisse für den Layer-2-Datenverkehr abzuleiten, welcher in den nächsten Szenarien nicht in Betracht gezogen wird.

Was für den Layer-2-Datenstrom aus dem UDP-Netzwerkverkehr abgeleitet werden kann, ist Paragraph «3.3.4.1» auf der vorherigen Seite zu entnehmen: Bei Übertragungen des Lasttyps «MIN» erzielt die Ethernet-Variante aus den im vorhin erwähnten Paragraph erläuterten Grund eine marginal höhere CPU- und Netzwerk-Last.

Aus den oben erwähnten Gründen werden folgende Datenverkehrströme für die nachfolgenden Szenarien verwendet:

Zeitspanne	**Ethernet** **MIN/MAX**	**TCP** **MIN/MAX**	**UDP** **MIN/MAX**
Ultrakurzzeit	-	✓	✓
Kurzzeit	-	✓	✓
Langzeit	-	-	-

Tabelle 3.42: Für die nachfolgenden Szenarien in Frage kommende Netzwerklasten

Dies ergibt folgende Netzwerklasten, welche, sofern es nicht anders erwähnt wird, in jedem nummerierten Szenario generiert werden (XX steht für die entsprechende Szenarionummer):

Bezeichnung	Übermitt-lungsart	Lasttyp	Zeitspanne	Dauer
XX.T.MIN.U	TCP	MIN	Ultrakurzzeit	60s
XX.T.MIN.K			Kurzzeit	5min
XX.T.MAX.U		MAX	Ultrakurzzeit	60s
XX.T.MAX.K			Kurzzeit	5min
XX.U.MIN.U	UDP	MIN	Ultrakurzzeit	60s
XX.U.MIN.K			Kurzzeit	5min
XX.U.MAX.U		MAX	Ultrakurzzeit	60s
XX.U.MAX.K			Kurzzeit	5min

Tabelle 3.43: Netzwerklasten eines Szenarios

Für jede generierte Netzwerklast werden pro Teilnehmer (z.B. shck-Client und shck-Server) folgende Statistik-Merkmale in den Resultaten (siehe Kapitel 4 auf Seite 104) aufgelistet:

- CPU-Last
  - CPU-Last [%] (jeweils Minimum, Durchschnitt und Maximum)
  - Systemtime [s] (jeweils Minimum, Durchschnitt und Maximum)
  - Usertime [s] (jeweils Minimum, Durchschnitt und Maximum)
- Netzwerkbelastung pro verwendetes Netzwerk-Interface (Normalfall: «prp1», «eth0», «eth1»)
  - RX-Bitrate (Download) [MBit/s] (jeweils Minimum, Durchschnitt und Maximum)
    - Wird nichts empfangen bzw. beträgt die RX-Bitrate konstant 0.00 (Beispiel: UDP-Traffic-Generierung betrachtet auf sendendem shck-Client auf PRP-Netzwerk-Interface), so wird dieses Merkmal für die entsprechende Netzwerklast nicht aufgeführt.
  - TX-Bitrate (Download) [MBit/s] (jeweils Minimum, Durchschnitt und Maximum)
    - Wird nichts versendet bzw. beträgt die TX-Bitrate konstant 0.00 (Beispiel: UDP-Traffic-Generierung betrachtet auf empfangendem shck-Server auf PRP-Netzwerk-Interface), so wird dieses Merkmal für die entsprechende Netzwerklast nicht aufgeführt.

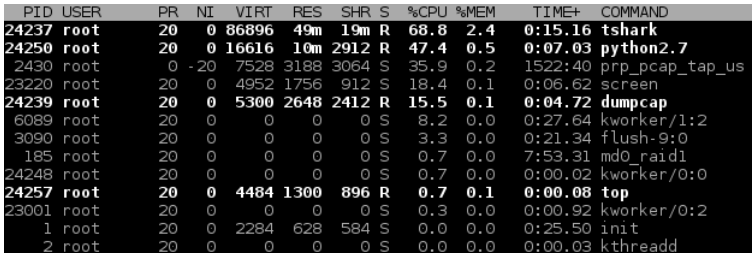
Bei der Netzwerkbelastung sind in einer PRP-Umgebung die RX-/TX-Bitraten bei den Netzwerk-Interfaces «eth0» und «eth1» nie konstant 0.00 MBit/s, da stets «PRP_Supervision»-Frames versendet werden, die vom Interface «prp1» nicht gesehen werden und direkt vom PRP-1 stack kommen (siehe Kapitel 2.1 auf Seite 14). Da die allein durch

«PRP_Supervision»-Frames erzielte Bitrate gerundet 0.00 MBit/s beträgt wird es in den Resultaten dieser Arbeit gleich behandelt wie wenn sie effektiv 0.00 MBit/s betragen würde.

3.3.4.2 Einfluss von TCP-Window-Size

Bei TCP-Netzwerklasten könnte es vorkommen, dass die TCP-Window-Size (siehe Kapitel 2.3 auf Seite 20) neben der CPU und der Netzwerkanbindung ein limitierender Faktor darstellt. Dazu werden 2 unterschiedliche TCP-Netzwerklasten mittels tshark betrachtet: Eine Last mit minimal und eine weitere Last mit maximal grossen TCP-Paketen, welche jeweils für eine Minute versendet wird. Diese Zeitspanne genügt, um bei konstantem Versand von TCP-Paketen aussagekräftige TCP-Window-Size-Werte zu erhalten.

Zu beachten ist, dass tshark die CPU erheblich beansprucht und somit ein Engpass beim Prozessor entstehen oder sich weiter verengen könnte, worunter die RX-/TX-Bitraten in Mitleidenschaft gezogen werden könnten.



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
24237	**root**	**20**	**0**	**86896**	**49m**	**19m**	**R**	**68.8**	**2.4**	**0:15.16**	**tshark**
24250	**root**	**20**	**0**	**16616**	**10m**	**2912**	**R**	**47.4**	**0.5**	**0:07.03**	**python2.7**
2430	root	0	-20	7528	3188	3064	S	35.9	0.2	1522:40	prp_pcap_tap_us
23220	root	20	0	4952	1756	912	S	18.4	0.1	0:06.62	screen
24239	**root**	**20**	**0**	**5300**	**2648**	**2412**	**R**	**15.5**	**0.1**	**0:04.72**	**dumpcap**
6089	root	20	0	0	0	0	S	8.2	0.0	0:27.64	kworker/1:2
3090	root	20	0	0	0	0	S	3.3	0.0	0:21.34	flush-9:0
185	root	20	0	0	0	0	S	0.7	0.0	7:53.31	md0_raid1
24248	root	20	0	0	0	0	S	0.7	0.0	0:00.02	kworker/0:0
24257	**root**	**20**	**0**	**4484**	**1300**	**896**	**R**	**0.7**	**0.1**	**0:00.08**	**top**
23001	root	20	0	0	0	0	S	0.3	0.0	0:00.92	kworker/0:2
1	root	20	0	2284	628	584	S	0.0	0.0	0:25.50	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.03	kthreadd

Abbildung 3.32: Ressourcen-Nutzung, dargestellt mit top (shck mit -t TCP -s MAX und tshark)

In den folgenden Grafiken ist ersichtlich, dass sich die TCP-Window-Size-Werte bei der Last mit minimalen Paketgrössen (Lasttyp «MIN») weder in den TCP-Paketen vom Sender noch vom Empfänger ändern. Das bedeutet, dass bei diesem Lasttyp die Performance nicht von der TCP-Window-Size limitiert wird.

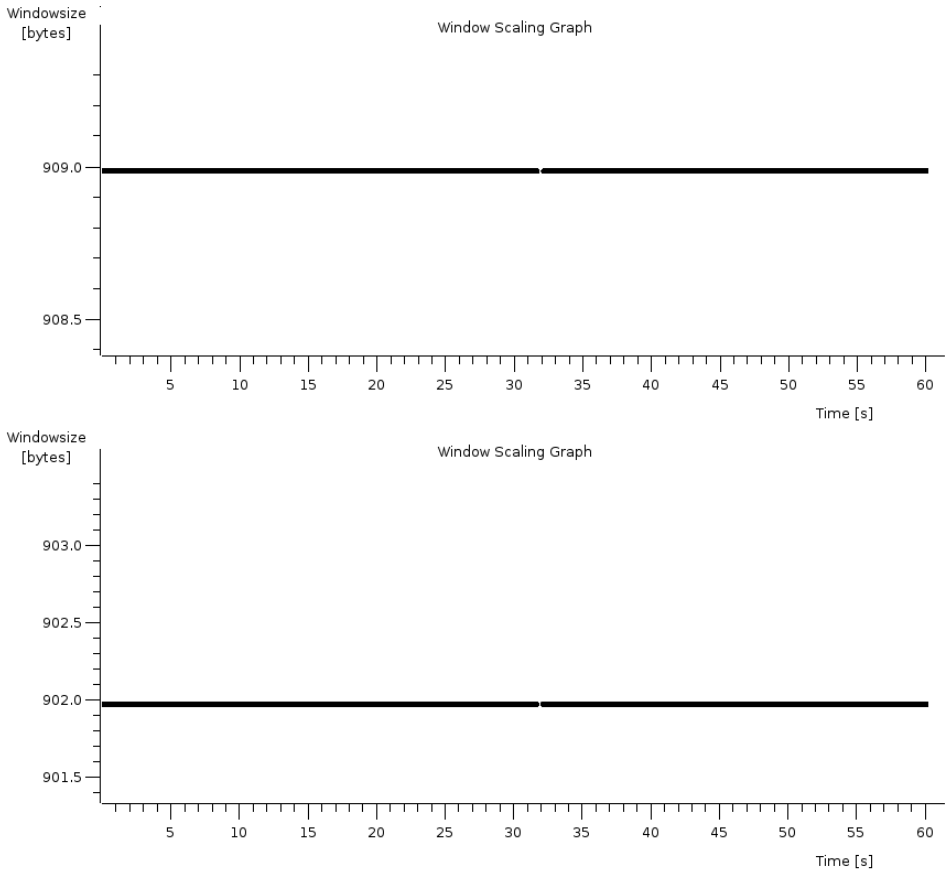


Abbildung 3.33: TCP-Window-Size, Lasttyp «MIN» (oben sendender Client, unten empfangender Server)

Beim Lasttyp «MAX» ist beim Client dasselbe wie beim Lasttyp «MIN» ersichtlich, jedoch ist beim Server zu Beginn eine Steigung festzustellen. Danach schwankt der Wert um das erreichte Maximum, fällt aber nie wieder auf den Anfangswert zurück. Wie beim Lasttyp «MIN» wird auch hier die Performance nicht von der TCP-Window-Size limitiert, da alle TCP-Window-Size-Werte nie auf 0 fallen und damit den Durchsatz beeinträchtigen würden.

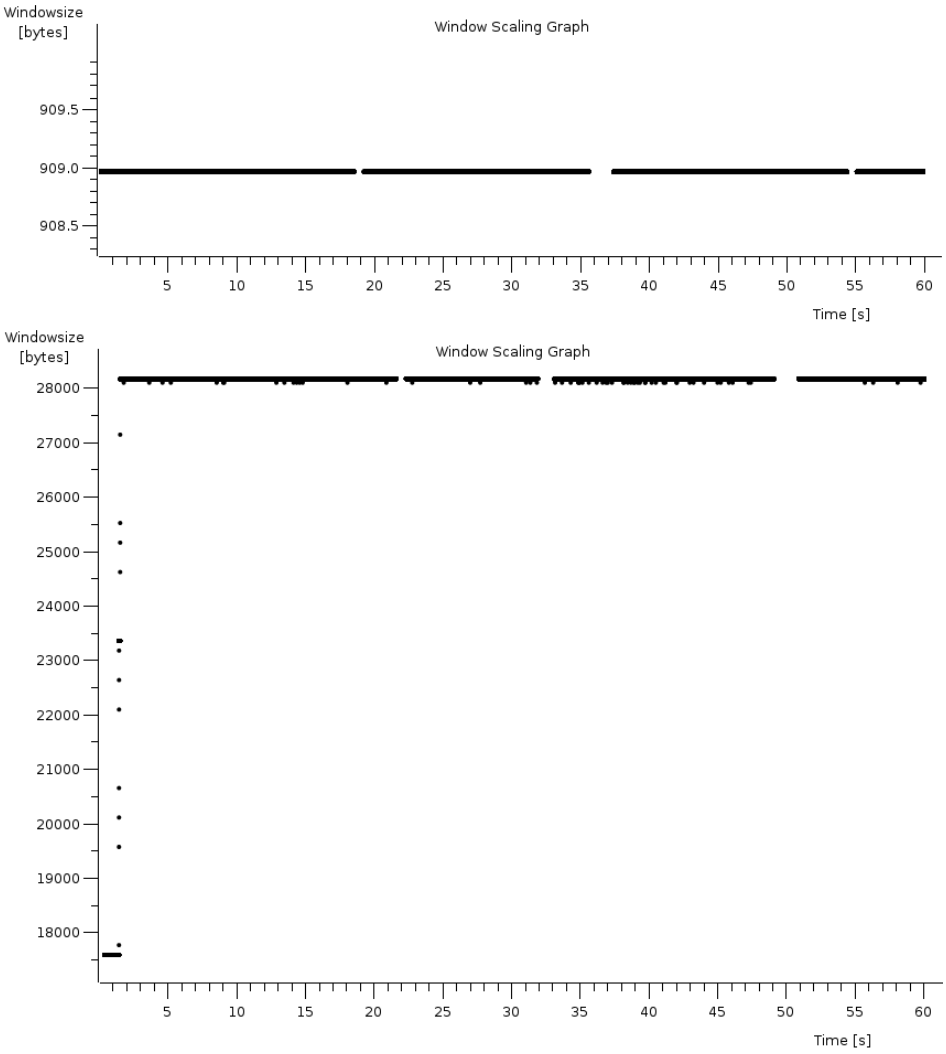


Abbildung 3.34: TCP-Window-Size, Lasttyp «MAX» (oben sendender Client, unten empfangender Server)

Ein weiterer Grund, dass die TCP-Window-Size kein limitierender Faktor ist, zeigen die Resultate von Szenario 01 (siehe Kapitel 3.3.4.3 auf der nächsten Seite) in Kapitel 4.1 auf Seite 104. Dort ist ersichtlich, dass beim Lasttyp «MAX» der theoretische Grenzwert (siehe Kapitel 3.3.1.3 auf Seite 77) erreicht wird und die Netzwerkanbindung der limitierende Faktor ist. Da bei diesem Lasttyp aufgrund der gegenüber dem Lasttyp «MIN» höher möglichen Durchsatzrate auch der TCP-Window-Size-Wert grösser ausfallen würde (siehe vorherige Abbildungen), kann es beim Lasttyp «MIN» nur die CPU oder ein anderer Faktor von TCP sein, der einen Engpass bildet.

Des Weiteren sind Lücken in den vorherigen Abbildungen festzustellen, welche bedeuten, dass zu diesem Zeitpunkt Wireshark kein Paket aufgezeichnet hat. Wären keine Pakete zu diesem Zeitpunkt übermittelt worden, würde bei der Ausgabe von meas ein Minimum von 0 MBit/s aufgezeigt werden, was jedoch nicht der Fall ist. Dies wird in der tshark-Aufzeichnung in Wireshark wie folgt aufgezeigt:

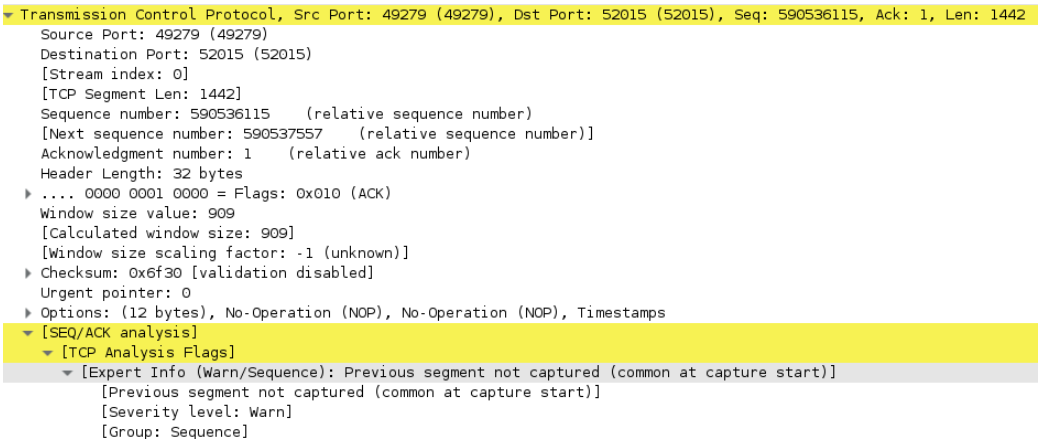


Abbildung 3.35: Wireshark-Meldung bei TCP-Netzwerklast: «Previous segment not captured»

Als Ursache dafür ist anzunehmen, dass tshark beim Aufzeichnen wegen dem Prozessor (hohe Lastbeanspruchung, die ebenfalls vom PRP-1 stack und shck benötigt wird) oder dem Festplattenspeicher an seine Grenzen gerät und er dem Netzwerkverkehr nicht mehr ganz folgen kann.

Schlussendlich kann man sagen, dass die TCP-Window-Size kein limitierender Faktor darstellt und TCP-Verkehr nicht durch die TCP-Window-Size beeinträchtigt werden kann. Nicht auszuschliessen sind jedoch andere Faktoren von TCP wie zum Beispiel das Warten des Senders auf eine Quittierung (ACK-Paket), die CPU oder die Netzwerkanbindung.

3.3.4.3 Szenario 01: Performance im PRP-Netzwerk

In diesem Szenario wird die ursprüngliche Konfiguration der Testumgebung (siehe Kapitel 3.1 auf Seite 23) verwendet, um die Ergebnisse dieses Szenarios als Refernzpunkt für weitere Messungen verwenden zu können.

Ablauf eines Szenarios	Seite 87
Generierte Netzwerklasten	Seite 95
Resultate und Interpretation	Seite 104

3.3.4.4 Szenario 02: Performance ohne PRP

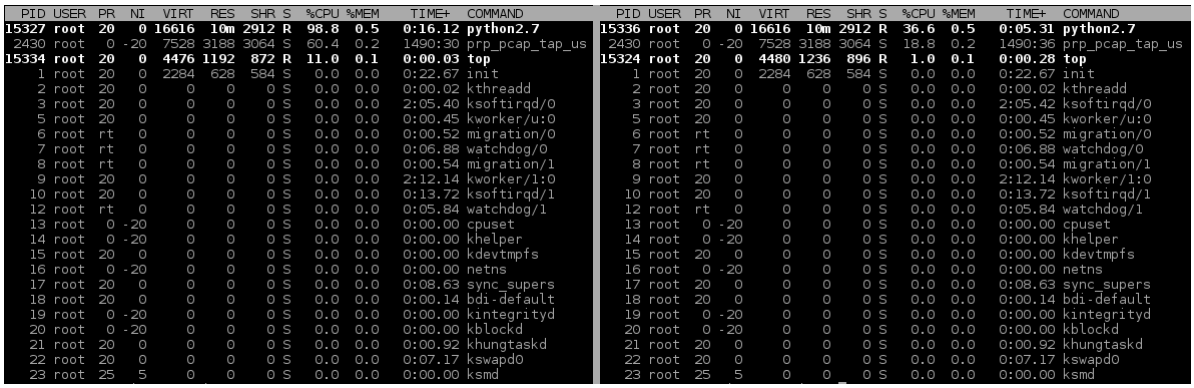
Die ursprüngliche Konfiguration (siehe Kapitel 3.1 auf Seite 23) wird für dieses Szenario wie folgt abgeändert:

- Der PRP-1 stack wird deaktiviert, woraus resultiert, dass es in diesem Szenario kein PRP-Netzwerkinterface (prp1) gibt und kein PRP in der Umgebung verwendet wird.

- Anstelle des PRP-Netzwerks wird nur das Netzwerk A verwendet.

Da der PRP-1 stack in diesem Szenario nicht in Betrieb ist, kann er nicht gemessen werden. Um die Performance-Unterschiede zwischen diesem und dem vorherigen Szenario (siehe Kapitel 3.3.4.3 auf der vorherigen Seite) ermitteln zu können, wird der Prozess zur Netzwerklastgenerierung (shck) und die Netzwerkbelastung mit und ohne PRP analysiert.

Während der Performance-Ermittlung sind der PRP-1 stack und shck die einzigen beiden Prozesse, welche signifikant Ressourcen beanspruchen. Indem in diesem Szenario shck anstelle des PRP-1 stacks gemessen wird, kann festgestellt werden, wie viel an Ressourcen für die PRP-Umgebung statt der Netzwerklastgenerierung verbraucht wird.



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
15327	**root**	**20**	**0**	**16616**	**10m**	**2912**	**R**	**98.8**	**0.5**	**0:16.12**	**python2.7**
2430	root	0	-20	7528	3188	3064	S	60.4	0.2	1490:30	prp_pcap_tap_us
15334	**root**	**20**	**0**	**4476**	**1192**	**872**	**R**	**11.0**	**0.1**	**0:00.03**	**top**
1	root	20	0	2284	628	584	S	0.0	0.0	0:22.67	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.02	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	2:05.40	ksoftirqd/0
5	root	20	0	0	0	0	S	0.0	0.0	0:00.45	kworker/u:0
6	root	rt	0	0	0	0	S	0.0	0.0	0:00.52	migration/0
7	root	rt	0	0	0	0	S	0.0	0.0	0:06.88	watchdog/0
8	root	rt	0	0	0	0	S	0.0	0.0	0:00.54	migration/1
9	root	20	0	0	0	0	S	0.0	0.0	2:12.14	kworker/1:0
10	root	20	0	0	0	0	S	0.0	0.0	0:13.72	ksoftirqd/1
12	root	rt	0	0	0	0	S	0.0	0.0	0:05.84	watchdog/1
13	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	cpuset
14	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	khelper
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
17	root	20	0	0	0	0	S	0.0	0.0	0:08.63	sync_supers
18	root	20	0	0	0	0	S	0.0	0.0	0:00.14	bdi-default
19	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
20	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
21	root	20	0	0	0	0	S	0.0	0.0	0:00.92	khungtaskd
22	root	20	0	0	0	0	S	0.0	0.0	0:07.17	kswapd0
23	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
15336	**root**	**20**	**0**	**16616**	**10m**	**2912**	**R**	**36.6**	**0.5**	**0:05.31**	**python2.7**
2430	root	0	-20	7528	3188	3064	S	18.8	0.2	1490:36	prp_pcap_tap_us
15324	**root**	**20**	**0**	**4480**	**1236**	**896**	**R**	**1.0**	**0.1**	**0:00.28**	**top**
1	root	20	0	2284	628	584	S	0.0	0.0	0:22.67	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.02	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	2:05.42	ksoftirqd/0
5	root	20	0	0	0	0	S	0.0	0.0	0:00.45	kworker/u:0
6	root	rt	0	0	0	0	S	0.0	0.0	0:00.52	migration/0
7	root	rt	0	0	0	0	S	0.0	0.0	0:06.88	watchdog/0
8	root	rt	0	0	0	0	S	0.0	0.0	0:00.54	migration/1
9	root	20	0	0	0	0	S	0.0	0.0	2:12.14	kworker/1:0
10	root	20	0	0	0	0	S	0.0	0.0	0:13.72	ksoftirqd/1
12	root	rt	0	0	0	0	S	0.0	0.0	0:05.84	watchdog/1
13	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	cpuset
14	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	khelper
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
17	root	20	0	0	0	0	S	0.0	0.0	0:08.63	sync_supers
18	root	20	0	0	0	0	S	0.0	0.0	0:00.14	bdi-default
19	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
20	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
21	root	20	0	0	0	0	S	0.0	0.0	0:00.92	khungtaskd
22	root	20	0	0	0	0	S	0.0	0.0	0:07.17	kswapd0
23	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd

Abbildung 3.36: Ressourcen-Nutzung, dargestellt mit top (links shck mit -t UDP -s MIN, rechts mit -t UDP -s MAX)

Die Resultate dieses Szenarios betragen die Performance-Werte von shck und die beanspruchte Netzwerklast, mal mit und mal ohne PRP.

Ablauf eines Szenarios Seite 87
Generierte Netzwerklasten Seite 95
Resultate und Interpretation Seite 113

3.3.4.5 Szenario 03: Effekt von Laufzeitunterscheidungen zwischen Netzwerk A und B

Asymmetrie A & B -> zusatzdelay

Interessanter fall: schwankender delay -> mal netz schneller und langsam -> oszillierend ECI via Python automatisieren? Paragon mit Variablem Delay dazwischen hängen

Ablauf eines Szenarios Seite 87
Generierte Netzwerklasten Seite 95
Resultate und Interpretation Seite 126

3.3.4.6 Szenario 04: Zeitweiser Ausfall eines Netzwerkpfades

In diesem Szenario wird die ursprüngliche Konfiguration der Testumgebung (siehe Kapitel 3.1 auf Seite 23) nicht modifiziert. Stattdessen wird die Netzwerkanbindung ans Netzwerk A über das Netzwerk-Interface «eth0» während der Messung nach einem bestimmten Intervall (z.B. alle 10 Sekunden) aktiviert / deaktiviert. Dadurch soll herausgefunden werden wie der PRP-1 stack mit zeitweisen Ausfällen umgeht. Dazu werden Messungen mit jeder Netzwerklast mit unterschiedlichen Intervallen sowie auch mit einem Komplettausfall durchgeführt, was in folgenden Fällen resultiert:

Fall	**Netzwerklast-Suffix / Fall**
Komplettausfall Netzwerk A	A
Intervall von 1s	1
Intervall von 2s	2
Intervall von 3s	3
Intervall von 4s	4
Intervall von 5s	5
Intervall von 6s	6
Intervall von 7s	7
Intervall von 8s	8
Intervall von 9s	9

Tabelle 3.44: Fälle von zeitweisem Ausfall eines Netzwerkpfades

Der oben erwähnte Netzwerklast-Suffix / Fall ist bei der Bezeichnung in den graphischen Resulaten auf Seite 128 der entsprechenden Netzwerklast angehängt und weist auf den verwendeten Intervall in Sekunden oder den Komplettausfall hin.

Es bestünde die Möglichkeit, die Netzwerk-Interfaces direkt über die Kommandozeile auf dem Host zu aktivieren / deaktivieren, jedoch werden dann die Nutzungsstatistiken des entsprechenden Interfaces im /proc-Dateisystem zurückgesetzt. Da diese Statistiken für die Messungen mit meas essenziell sind, stellt dieses Vorgehen hier keine Option dar. Aus diesem Grund werden bei dem Host «srv01» ECI-Boxen (siehe Tabelle 3.20 auf Seite 39) bei den Netzwerk-Interfaces «eth0» und «eth1» dazwischen geschaltet. Diese werden bei beiden Interfaces dazwischen geschaltet, damit die beiden Netzwerkanbindungen äquivalent zueinander bleiben (selbe Distanz / Verzögerung).

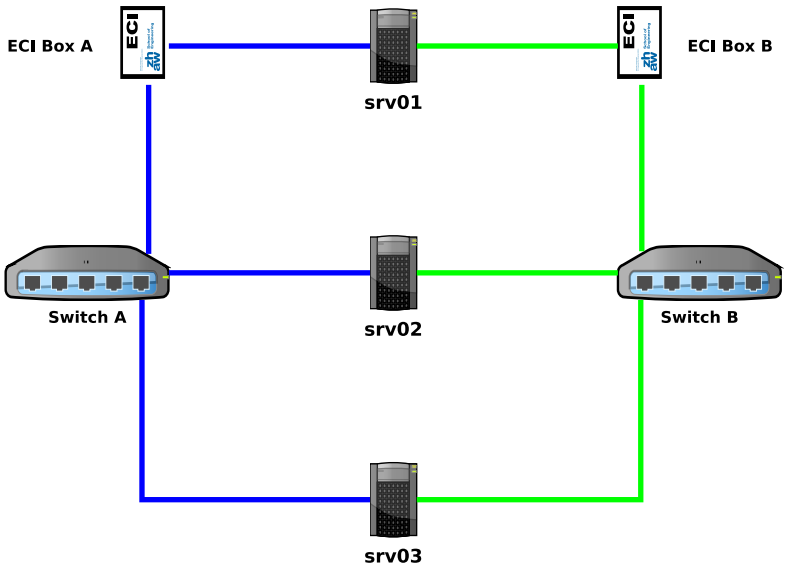


Abbildung 3.37: PRP-Netzwerkaufbau mit ECI-Boxen

Eine ECI-Box verursacht auf einem Netzwerkpfad einen Delay von $1.144 \pm 6\mu s$, was vernachlässigt werden kann, da dieser Delay bei beiden Netzwerkanbindungen auftritt und weil die versendeten Pakete zwar um diesen Delay später ankommen, jedoch im Steady State gemessen wird, in welchem ständig die selbe Last versendet wird.

Gesteuert werden die ECI-Boxen, welche am Host «GATEWAY PC» (siehe Kapitel 3.1.3 auf Seite 29) via USB angeschlossen sind, über Python-Scripts. Auf diesem Host wird in diesem Szenario ein Bash-Skript ausgeführt, welches das Python-Skript zur ECI-Box-Steuerung ausführt und auf dem Host «srv03» via SSH über das Netzwerk Ext die Messung auf «srv01» und «srv02» startet. Hier wird lediglich die ECI-Box am Interface «eth0» des Hosts «srv01» gesteuert, während die zweite ECI-Box im aktiven Zustand bleibt.

Ablauf eines Szenarios	Seite 87
Generierte Netzwerklasten	Seite 95
Resultate und Interpretation	Seite 126

3.3.4.7 Szenario 05: Abhängigkeit vom verwendeten Protokoll

Innerhalb dieses Szenarios werden keine Messungen vorgenommen, sondern die Resultate aus Szenario 01 (siehe Kapitel 3.3.4.3 auf Seite 99) für den Vergleich zwischen TCP- und UDP-Netzwerklasten übernommen. Die Resultate und Interpretation zum Vergleich TCP/UDP sind in Kapitel 4.5 auf Seite 138 aufgeführt.

Für den Vergleich zwischen reiner Layer-2- und UDP-Netzwerklast kann das Kapitel 3.3.4.1 auf Seite 90 konsultiert werden.

Ablauf eines Szenarios	Seite 87
Generierte Netzwerklasten	Seite 95
Resultate und Interpretation	Seite 138

3.3.4.8 Szenario 06: Einfluss nicht entfernter Duplikate auf Funktion und Performance

Delay Ankunft zwischen Original und Duplikat zu gross

Möglich? Neue SeqNr hat gleiche wie alte wo noch in PRP Tabelle ist

Kann Speicher an Duplikaten im Stack komplett ausgelastet werden?

Wie geht stack vor wenn Duplikattabelle voll? Wenn zb LAN_A aus? wird ältestes überschrieben?

Ablauf eines Szenarios	Seite 87
Generierte Netzwerklasten	Seite 95
Resultate und Interpretation	Seite 140

3.3.4.9 Szenario 07: Auswirkungen, wenn Frames out-of-sequence ankommen

Out of sequence erzwingen (Langes zuerst, dann kurzes -> kurzes kommt vor langem an)

Ablauf eines Szenarios	Seite 87
Generierte Netzwerklasten	Seite 95
Resultate und Interpretation	Seite 140

3.3.4.10 Szenario 08: Einluss von Offload-Mechanismen

Ablauf eines Szenarios	Seite 87
Generierte Netzwerklasten	Seite 95
Resultate und Interpretation	Seite 140

4 Resultate und Interpretation

4.1 Szenario 01: Performance im PRP-Netzwerk

4.1.1 shck-Client

Name	CPU-Last [%]			Systemtime [s]			Usertime [s]		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
01.U.MIN.U	62.99	63.91	65.00	0.54	0.57	0.60	0.04	0.07	0.10
01.U.MIN.K	55.99	57.37	59.00	0.45	0.50	0.55	0.03	0.07	0.12
01.U.MAX.U	16.00	17.12	18.00	0.14	0.16	0.17	0.00	0.02	0.04
01.U.MAX.K	16.00	17.12	18.00	0.12	0.15	0.18	0.00	0.02	0.05
01.T.MIN.U	32.00	33.26	34.00	0.26	0.29	0.31	0.02	0.04	0.07
01.T.MIN.K	32.00	33.40	35.00	0.24	0.29	0.32	0.01	0.05	0.09
01.T.MAX.U	39.00	40.98	42.00	0.33	0.36	0.39	0.03	0.05	0.09
01.T.MAX.K	39.00	40.63	42.00	0.31	0.35	0.38	0.02	0.05	0.10

Tabelle 4.1: Resultat Szenario 01 auf shck-Client: CPU-Last, System- / Usertime des PRP-1 stacks

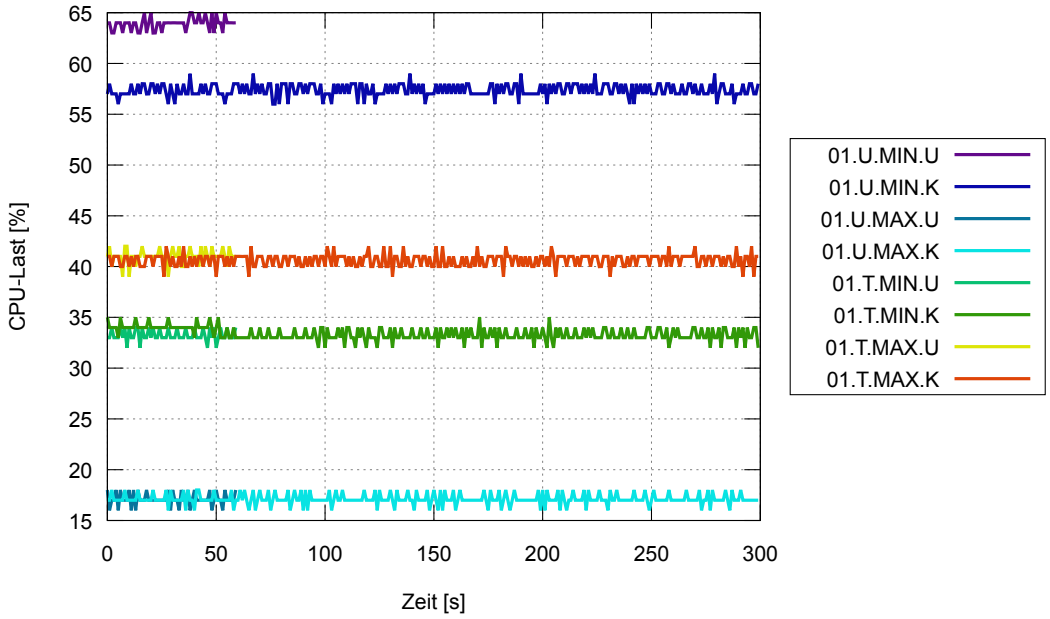


Abbildung 4.1: Resultat Szenario 01 auf shck-Client: CPU-Last des PRP-1 stacks

Name	RX-Bitrate [MBit/s]								
	prp1			eth0			eth1		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
01.T.MIN.U	1.77	1.77	1.78	2.03	2.04	2.05	2.03	2.04	2.05
01.T.MIN.K	1.74	1.78	1.79	2.00	2.05	2.06	2.00	2.05	2.06
01.T.MAX.U	2.15	2.15	2.16	2.48	2.48	2.48	2.48	2.48	2.48
01.T.MAX.K	2.13	2.15	2.16	2.45	2.48	2.49	2.45	2.48	2.49

Tabelle 4.2: Resultat Szenario 01 auf shck-Client: RX-Bitrate [MBit/s]

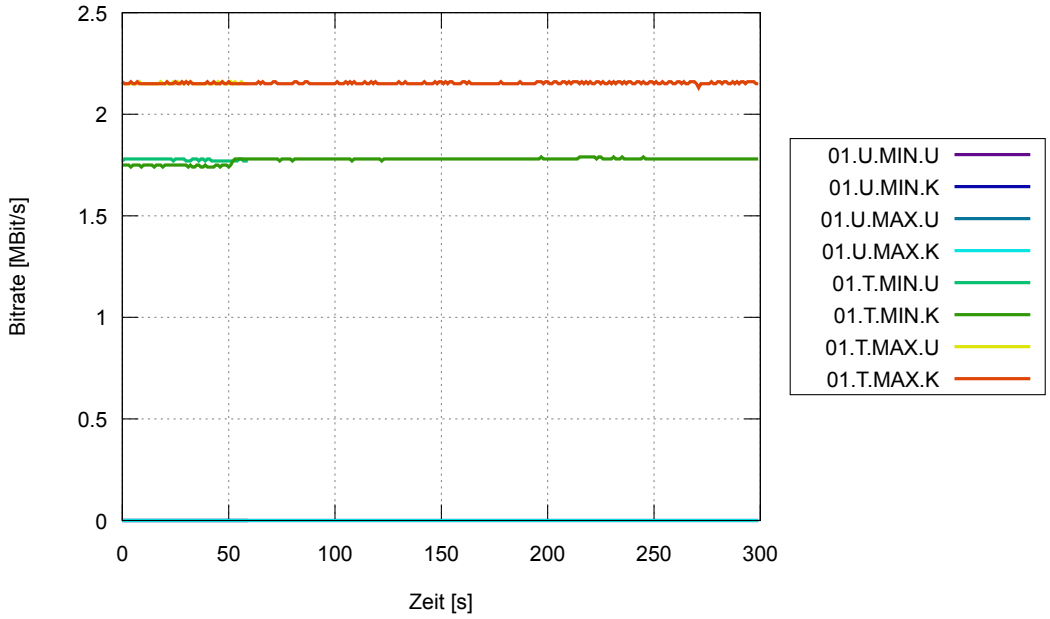


Abbildung 4.2: Resultat Szenario 01 auf shck-Client: RX-Bitrate auf «prp1»-Netzwerk-Interface [MBit/s]

Name	TX-Bitrate [MBit/s]								
	prp1			eth0			eth1		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
01.U.MIN.U	7.07	7.22	7.30	8.25	8.43	8.52	8.25	8.43	8.52
01.U.MIN.K	6.68	6.88	7.00	7.80	8.03	8.17	7.80	8.03	8.17
01.U.MAX.U	97.69	98.05	98.42	98.64	98.71	98.72	98.66	98.71	98.74
01.U.MAX.K	97.67	98.06	98.47	98.66	98.71	98.73	98.66	98.71	98.72
01.T.MIN.U	1.94	1.95	1.95	2.21	2.22	2.23	2.21	2.22	2.22
01.T.MIN.K	1.91	1.93	1.94	2.17	2.20	2.21	2.17	2.20	2.21
01.T.MAX.U	98.02	98.05	98.08	98.65	98.70	98.73	98.65	98.70	98.73
01.T.MAX.K	98.01	98.05	98.10	98.65	98.70	98.76	98.65	98.70	98.74

Tabelle 4.3: Resultat Szenario 01 auf shck-Client: TX-Bitrate [MBit/s]

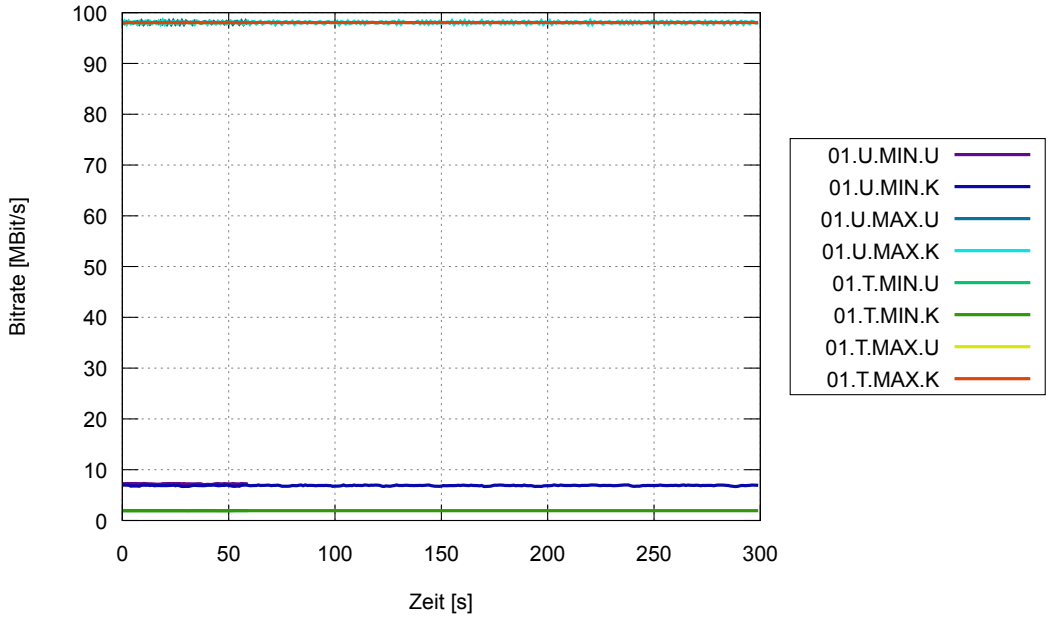


Abbildung 4.3: Resultat Szenario 01 auf shck-Client: TX-Bitrate auf «prp1»-Netzwerk-Interface [MBit/s]

4.1.2 shck-Server

Name	CPU-Last [%]			Systemtime [s]			Usertime [s]		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
01.U.MIN.U	62.99	70.76	72.99	0.55	0.60	0.66	0.06	0.11	0.14
01.U.MIN.K	61.99	65.61	68.99	0.49	0.56	0.62	0.04	0.10	0.17
01.U.MAX.U	36.00	44.95	54.00	0.29	0.38	0.48	0.03	0.07	0.10
01.U.MAX.K	36.00	43.89	55.00	0.27	0.37	0.50	0.02	0.07	0.11
01.T.MIN.U	27.00	28.15	29.00	0.20	0.24	0.26	0.02	0.04	0.08
01.T.MIN.K	27.00	28.42	30.00	0.21	0.24	0.28	0.01	0.04	0.08
01.T.MAX.U	55.00	63.36	69.00	0.45	0.55	0.64	0.03	0.09	0.14
01.T.MAX.K	55.00	62.31	68.00	0.42	0.54	0.63	0.01	0.08	0.14

Tabelle 4.4: Resultat Szenario 01 auf shck-Server: CPU-Last, System- / Usertime des PRP-1 stacks

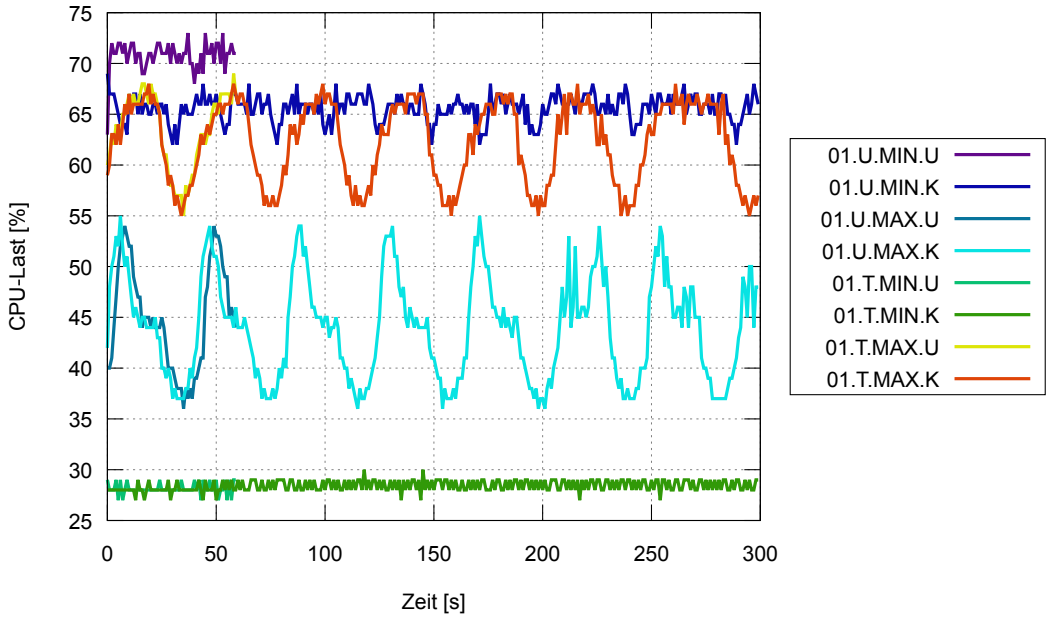


Abbildung 4.4: Resultat Szenario 01 auf shck-Server: CPU-Last des PRP-1 stacks

Name	RX-Bitrate [MBit/s]								
	prp1			eth0			eth1		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
01.U.MIN.U	6.61	7.21	7.30	7.71	8.41	8.51	7.71	8.41	8.51
01.U.MIN.K	6.68	6.88	7.00	7.79	8.03	8.16	7.79	8.03	8.16
01.U.MAX.U	98.03	98.05	98.06	98.64	98.70	98.75	98.64	98.70	98.75
01.U.MAX.K	98.03	98.05	98.08	98.64	98.70	98.75	98.64	98.70	98.75
01.T.MIN.U	1.94	1.95	1.96	2.21	2.22	2.23	2.21	2.22	2.23
01.T.MIN.K	1.88	1.93	1.94	2.14	2.20	2.21	2.14	2.20	2.21
01.T.MAX.U	98.02	98.04	98.06	98.67	98.70	98.75	98.67	98.70	98.75
01.T.MAX.K	98.02	98.05	98.08	98.63	98.70	98.75	98.63	98.70	98.75

Tabelle 4.5: Resultat Szenario 01 auf shck-Server: RX-Bitrate [MBit/s]

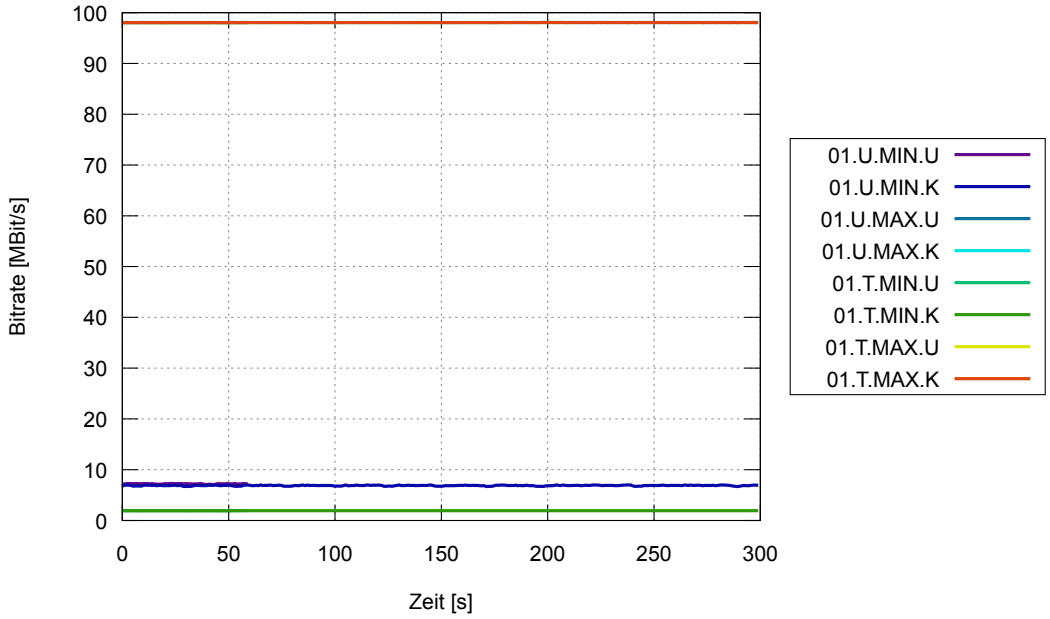


Abbildung 4.5: Resultat Szenario 01 auf shck-Server: RX-Bitrate auf «prp1»-Netzwerk-Interface [MBit/s]

Name	TX-Bitrate [MBit/s]								
	prp1			eth0			eth1		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
01.T.MIN.U	1.77	1.77	1.78	2.03	2.04	2.05	2.03	2.04	2.05
01.T.MIN.K	1.72	1.78	1.79	1.98	2.04	2.06	1.98	2.04	2.06
01.T.MAX.U	2.15	2.15	2.16	2.48	2.48	2.48	2.48	2.48	2.48
01.T.MAX.K	2.12	2.15	2.16	2.44	2.48	2.49	2.44	2.48	2.49

Tabelle 4.6: Resultat Szenario 01 auf shck-Server: TX-Bitrate [MBit/s]

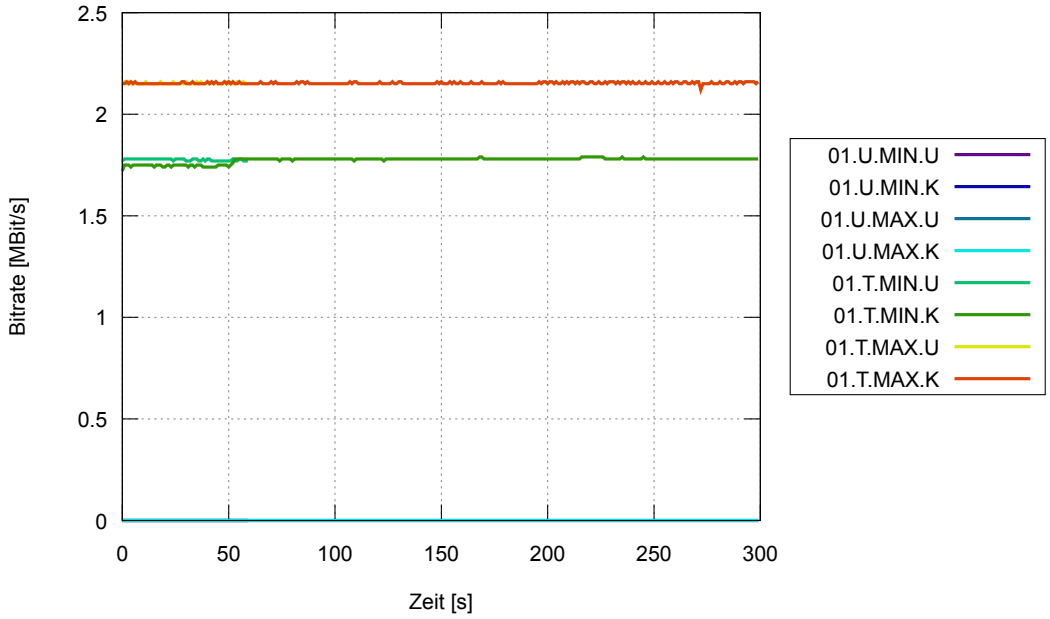


Abbildung 4.6: Resultat Szenario 01 auf shck-Server: TX-Bitrate auf «prp1»-Netzwerk-Interface [MBit/s]

4.1.3 Interpretation

Die Resultate dieses Szenarios weisen bei den Netzwerklasten innerhalb der selben Übermittlungsart und des selben Lasttyps (sie unterscheiden sich nur in der Zeitspanne) keine markanten Unterschiede auf, was die Reproduzierbarkeit der Messungen beweist. Des Weiteren wird aus dieser Erkenntnis in diesem Kapitel nicht zwischen den Zeitspannen unterschieden.

4.1.3.1 Limitierungen / Engpässe

Die Netzwerklasten mit Lasttyp «MAX» erreichen die theoretischen Grenzwerte (siehe Kapitel 3.3.1.3 auf Seite 77) und werden somit von der Netzwerkanbindung limitiert. Anhand der ermittelten Werte kann festgestellt werden, dass bei diesem Lasttyp die TCP-Datenströme im Vergleich zu UDP eine höhere CPU-Last verursachen, jedoch die selben TX-Bitraten auf dem shck-Client und RX-Bitraten auf dem shck-Server erzielen (theoretischer Grenzwert). Eine weitere Abweichung dieser beiden Strömen mit selbem Lasttyp ist, dass der shck-Client bei TCP-Messungen Quittierungen und bei UDP-Messungen keine Antwort vom shck-Server erhält, was die höhere CPU-Last bei TCP erklärt.

Beim Lasttyp «MIN» ist zwischen dem TCP- und UDP-Verkehr ein grosser Unterschied festzustellen. Ihre Gemeinsamkeit beträgt aber darin, dass beide Ströme nicht den theoretischen Grenzwert erlangen. Bei der TCP-Last fällt hier jedoch aufgrund der Natur

von TCP zusätzlich das Warten auf Quittierungen vom Server an, weshalb hier im Gegensatz zur UDP-Last nicht permanent versendet werden kann. Somit erzielt bei diesem Lasttyp der UDP-Datenstrom eine höhere CPU-Last, kann jedoch durch den häufigeren Versand von Paketen auf dem Client eine höhere TX-Bitrate erzielen. Daraus lässt sich schliessen, dass der UDP-Verkehr des Lasttyps «MIN» von der CPU limitiert wird. Beim TCP-Verkehr des selben Lasttyps stösst man auf einen Engpass, der sich durch die Wartezeit auf TCP-Quittierungen ergibt.

Zusammengefasst ergeben sich folgende Engpässe:

- UDP mit Lasttyp «MIN»: Engpass durch Prozessor
- UDP mit Lasttyp «MAX»: Limitierung durch Netzwerkanbindung
- TCP mit Lasttyp «MIN»: Einschränkung durch TCP (Warten auf Quittierungen vom Empfänger)
- TCP mit Lasttyp «MAX»: Limitierung durch Netzwerkanbindung

4.1.3.2 Verhalten zwischen shck-Client und -Server

Bezüglich der Belastung des Prozessors durch den PRP-1 stack ist pro Übertragungs- und Lasttyp folgende absteigende Reihenfolge ersichtlich:

shck-Client

- UDP mit Lasttyp «MIN» höchste CPU-Last
- TCP mit Lasttyp «MAX»
- TCP mit Lasttyp «MIN»
- UDP mit Lasttyp «MAX» kleinste CPU-Last

shck-Server

- UDP mit Lasttyp «MIN» höchste CPU-Last
- TCP mit Lasttyp «MAX»
- UDP mit Lasttyp «MAX»
- TCP mit Lasttyp «MIN» kleinste CPU-Last

Den Resultaten ist des Weiteren zu entnehmen, das bei den Netzwerklasten über UDP (besonders mit dem hier vom Netzwerk limitierten Lasttyp «MAX») auf dem shck-Server mehr CPU-Last verursacht wird. Dies bedeutet, dass der PRP-1 stack für das Empfangen mehr Leistung beansprucht als für das Senden.

Für Messungen mit TCP ist zu beachten, dass der PRP-1 stack in solchen Fällen TCP-Pakete sendet und Quittierungen vom Server erhält. Hier beansprucht lediglich der Lasttyp «MAX» mehr Rechenleistung vom Server als vom Client.

Bei der Netzwerklast via TCP mit Lasttyp «MIN» ist die Differenz zwischen Client und Server zwar weniger markant als bei den anderen Netzwerklasten, jedoch werden hier weniger Pakete versandt als via UDP mit dem Lasttyp «MIN». Dies resultiert aus der Tatsache, dass bei TCP der Client auf Quittierungen wartet anstelle konstant Pakete zu

senden, was eine geringere CPU-Last und kleinere Bitraten ergibt. Aus diesem Grund empfängt der PRP-1 stack bei dieser Netzwerklast weniger Pakete als bei den anderen Datenströmen und beansprucht somit auch weniger Ressourcen, was die folgende Rechnung mit den ermittelten Werten (durschnittliche TX-Bitrate von shck-Client auf Netzwerk-Interface «eth0») aufzeigt:

$$Pakete_{TCP:MIN} = \left\lceil \frac{2.22\,MBit/s}{77\,Byte} \right\rceil = 3'603\,Pakete$$

$$Pakete_{TCP:MAX} = \left\lceil \frac{98.7\,MBit/s}{1518\,Byte} \right\rceil = 8'128\,Pakete$$

$$Pakete_{UDP:MAX} = \left\lceil \frac{98.7\,MBit/s}{1518\,Byte} \right\rceil = 8'128\,Pakete$$

$$Pakete_{UDP:MIN} = \left\lceil \frac{8.43\,MBit/s}{70\,Byte} \right\rceil = 15'054\,Pakete$$

Bezüglich der durchschnittlich beanspruchten Netzwerklast ist ersichtlich, dass Client und Server dieselben Bitraten erzielen. Die RX-Bitraten des Clients entsprechen den TX-Bitraten des Servers so wie auch die TX-Bitraten des Clients zu den RX-Bitraten des Clients.

Letzten Endes unterscheiden sich die Messwerte von Client und Server lediglich in der beanspruchten CPU-Last.

4.1.3.3 Verhalten des PRP-1 stacks

Die ermittelten Messwerte zeigen auf, dass sich der PRP-1 stack durchgehend mehr im Kernelspace (Systemtime) als im Userspace (Usertime) befindet. Somit beschäftigt sich der PRP-1 stack häufiger mit dem Betriebssystem-Funktionen als mit den administrativen Aufgaben des PRP-Protokolls.

Zu den Tätigkeiten im Kernel-/Userspace gehören u.a.:

- Kernelspace
  - Kommunikation von und zum virtuellen Netzwerk-Interface «prp1» bzw. Datenströme mit Hardware austauschen
  - Handhabung von Timern und Interrupts
- Userspace
  - Anfügen und Entfernen des RCTs am und vom Frame
  - Organisation der Tabelle zur Duplikaterkennung (Aging, Einträge hinzufügen / löschen)

PROFILING WIP

4.1.3.4 Weitere Aspekte

Eine weitere Beobachtung kann bei den Schwankungsbändern (Differenz von Maximal- und Minimal-Wert) der unterschiedlichen Statistik-Merkmalen gemacht werden. Im Vergleich zur Steady-State-Untersuchung in Kapitel 3.3.1.4 auf Seite 78 fallen diese kleiner aus. Dies ist auf den Einfluss der gewählten Messintervallgrössen zurück zu führen. Bei der Steady-State-Untersuchung selektierte man ein Intervall von 0.1s, der Allgemeinfall, wie er u.a. in diesem Szenario angewandt wird, beträgt jedoch 1s. Die daraus entstehenden Konsequenzen werden in Kapitel 3.3.2 auf Seite 85 erläutert.

4.2 Szenario 02: Performance ohne PRP

4.2.1 Ohne PRP

4.2.1.1 shck-Client

Name	CPU-Last [%]			Systemtime [s]			Usertime [s]		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
02.U.MIN.U	98.99	99.99	100.99	0.11	0.16	0.23	0.77	0.84	0.89
02.U.MIN.K	98.98	100.00	100.99	0.07	0.14	0.21	0.79	0.86	0.93
02.U.MAX.U	32.00	32.63	34.00	0.03	0.06	0.09	0.24	0.27	0.30
02.U.MAX.K	31.00	32.22	34.00	0.02	0.06	0.10	0.22	0.27	0.30
02.T.MIN.U	98.99	99.97	100.99	0.13	0.20	0.29	0.72	0.80	0.87
02.T.MIN.K	98.98	100.00	100.99	0.05	0.11	0.23	0.76	0.89	0.95
02.T.MAX.U	34.00	35.11	36.00	0.05	0.08	0.12	0.23	0.27	0.30
02.T.MAX.K	34.00	35.53	37.00	0.04	0.08	0.13	0.23	0.28	0.32

Tabelle 4.7: Resultat Szenario 02 auf shck-Client ohne PRP: CPU-Last, System- / Usertime der shck-Applikation

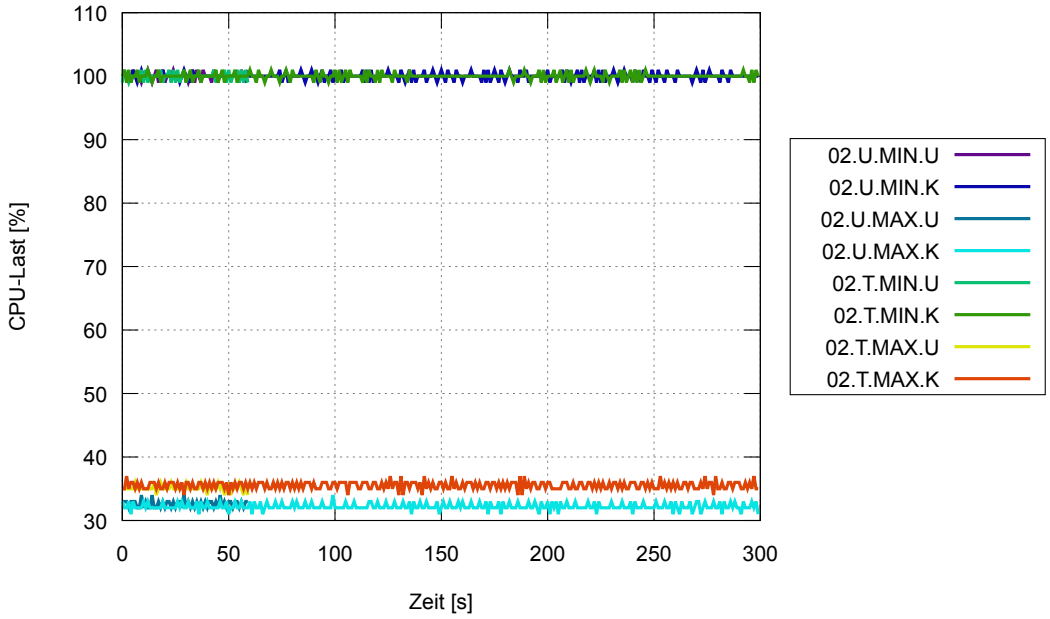


Abbildung 4.7: Resultat Szenario 02 auf shck-Client ohne PRP: CPU-Last der shck-Applikation

Bezeichnung	RX-Bitrate [MBit/s]		
	eth0		
	Min	Avg	Max
02.T.MIN.U	4.04	4.07	4.09
02.T.MIN.K	3.78	3.81	4.06
02.T.MAX.U	2.28	2.28	2.28
02.T.MAX.K	2.28	2.28	2.39

Tabelle 4.8: Resultat Szenario 02 auf shck-Client ohne PRP: RX-Bitrate [MBit/s]

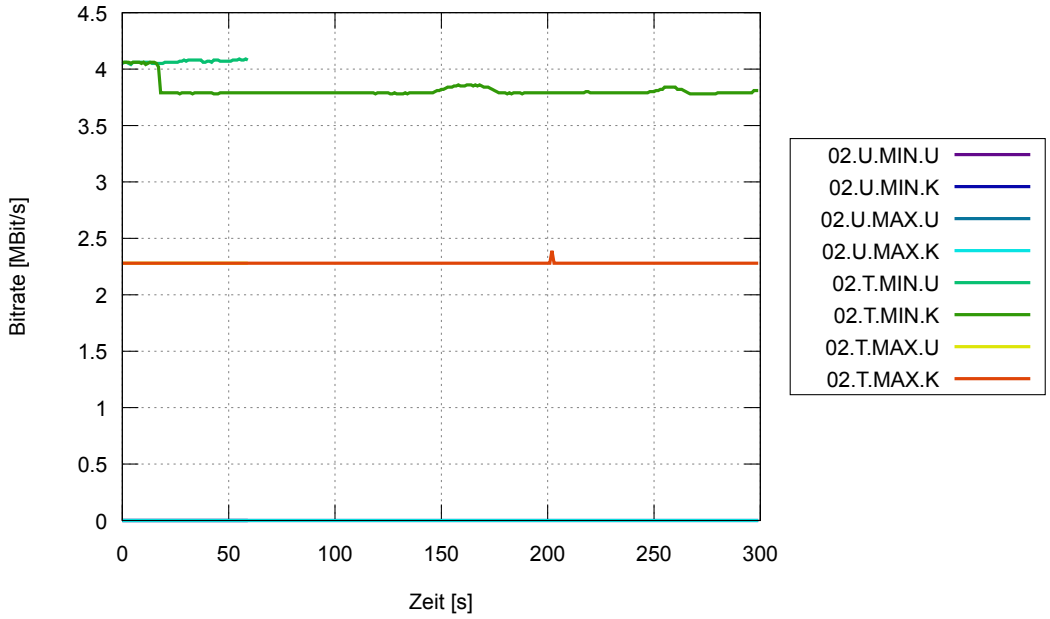


Abbildung 4.8: Resultat Szenario 02 auf shck-Client ohne PRP: RX-Bitrate auf «eth0»-Netzwerk-Interface [MBit/s]

Bezeichnung	TX-Bitrate [MBit/s]		
	eth0		
	Min	Avg	Max
02.U.MIN.U	13.28	13.34	13.38
02.U.MIN.K	13.44	13.53	13.57
02.U.MAX.U	98.66	98.71	98.72
02.U.MAX.K	98.66	98.71	98.74
02.T.MIN.U	4.23	4.25	4.28
02.T.MIN.K	3.98	4.01	4.25
02.T.MAX.U	98.66	98.71	98.72
02.T.MAX.K	97.14	98.70	98.74

Tabelle 4.9: Resultat Szenario 02 auf shck-Client ohne PRP: TX-Bitrate [MBit/s]

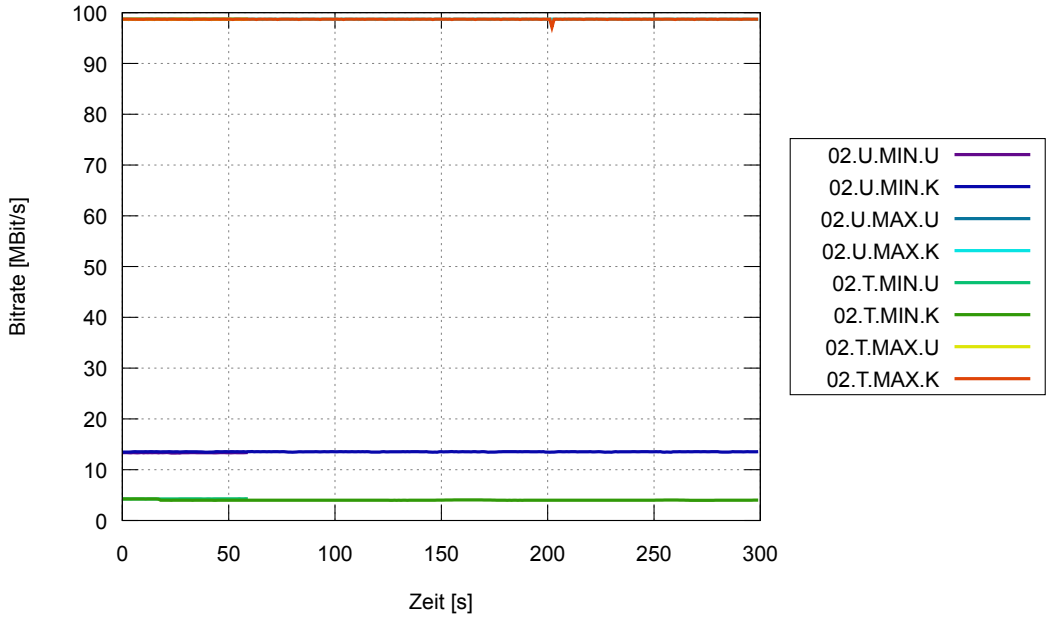


Abbildung 4.9: Resultat Szenario 02 auf shck-Client ohne PRP: TX-Bitrate auf «eth0»-Netzwerk-Interface [MBit/s]

4.2.1.2 shck-Server

Name	CPU-Last [%]			Systemtime [s]			Usertime [s]		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
02.U.MIN.U	58.99	61.35	62.00	0.16	0.21	0.27	0.35	0.41	0.46
02.U.MIN.K	57.99	62.34	64.00	0.14	0.21	0.30	0.32	0.41	0.47
02.U.MAX.U	19.00	20.15	22.00	0.04	0.08	0.11	0.09	0.13	0.16
02.U.MAX.K	19.00	20.15	21.00	0.04	0.07	0.11	0.09	0.13	0.16
02.T.MIN.U	16.00	17.05	18.00	0.03	0.09	0.12	0.04	0.08	0.14
02.T.MIN.K	15.00	16.03	18.00	0.05	0.09	0.12	0.04	0.07	0.11
02.T.MAX.U	18.00	18.83	20.00	0.07	0.10	0.14	0.05	0.09	0.12
02.T.MAX.K	17.99	18.85	20.00	0.07	0.10	0.15	0.04	0.08	0.12

Tabelle 4.10: Resultat Szenario 02 auf shck-Server ohne PRP: CPU-Last, System- / Usertime der shck-Applikation

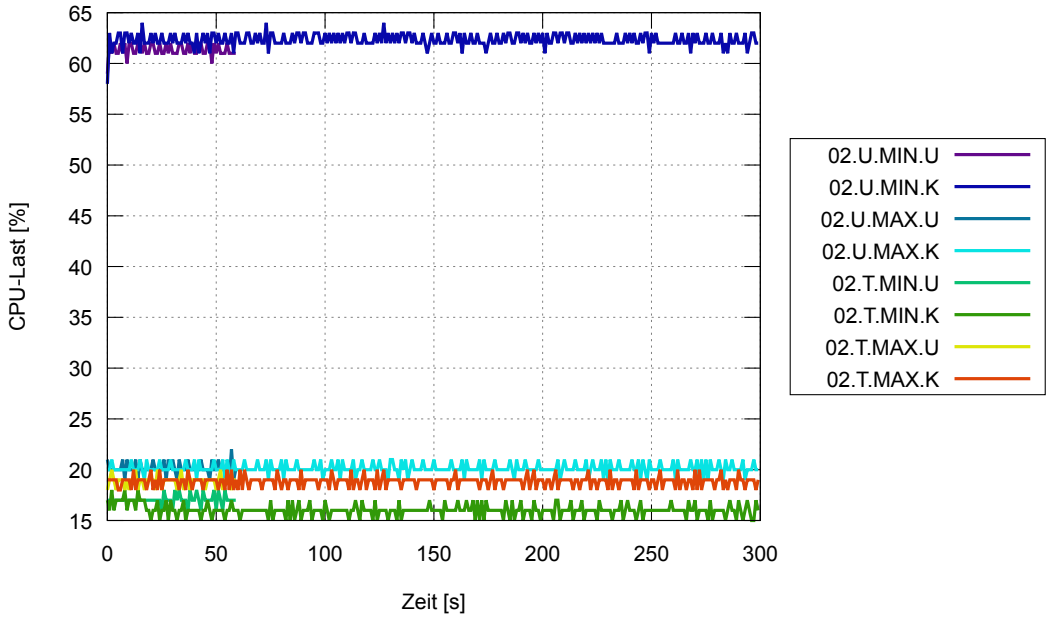


Abbildung 4.10: Resultat Szenario 02 auf shck-Server ohne PRP: CPU-Last der shck-Applikation

Bezeichnung	**RX-Bitrate [MBit/s]**		
	eth0		
	Min	**Avg**	**Max**
02.U.MIN.U	12.73	13.33	13.38
02.U.MIN.K	12.80	13.53	13.58
02.U.MAX.U	98.69	98.70	98.75
02.U.MAX.K	98.64	98.70	98.75
02.T.MIN.U	4.23	4.25	4.28
02.T.MIN.K	3.98	4.01	4.25
02.T.MAX.U	98.68	98.70	98.75
02.T.MAX.K	98.61	98.70	98.76

Tabelle 4.11: Resultat Szenario 02 auf shck-Server ohne PRP: RX-Bitrate [MBit/s]

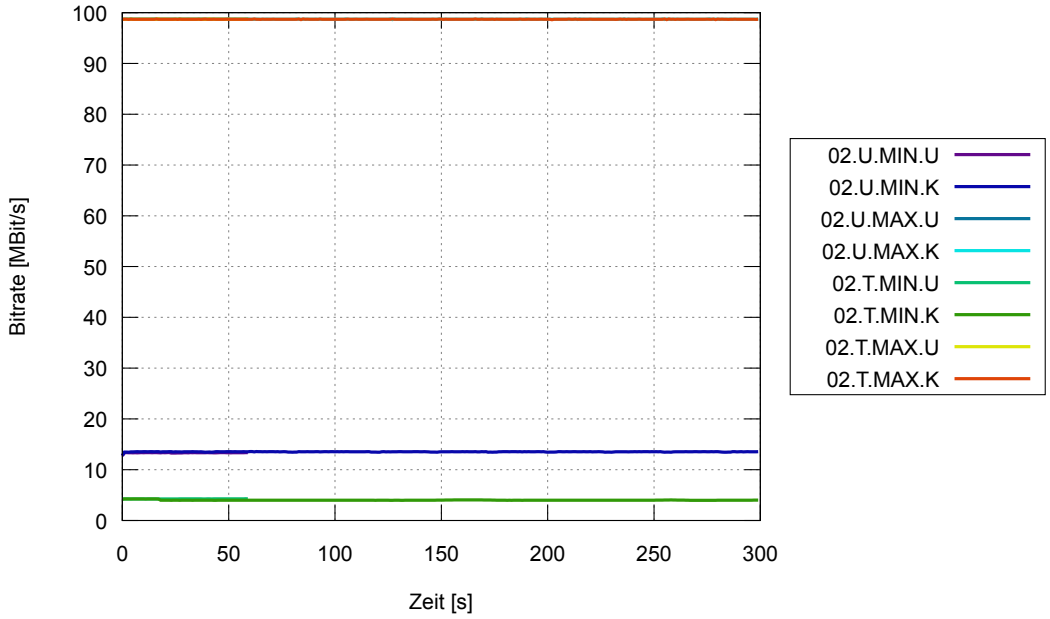


Abbildung 4.11: Resultat Szenario 02 auf shck-Server ohne PRP: RX-Bitrate auf «eth0»-Netzwerk-Interface [MBit/s]

Bezeichnung	**TX-Bitrate [MBit/s]**		
	eth0		
	Min	**Avg**	**Max**
02.T.MIN.U	4.04	4.07	4.09
02.T.MIN.K	3.78	3.81	4.06
02.T.MAX.U	2.28	2.28	2.28
02.T.MAX.K	2.28	2.28	2.39

Tabelle 4.12: Resultat Szenario 02 auf shck-Server ohne PRP: TX-Bitrate [MBit/s]

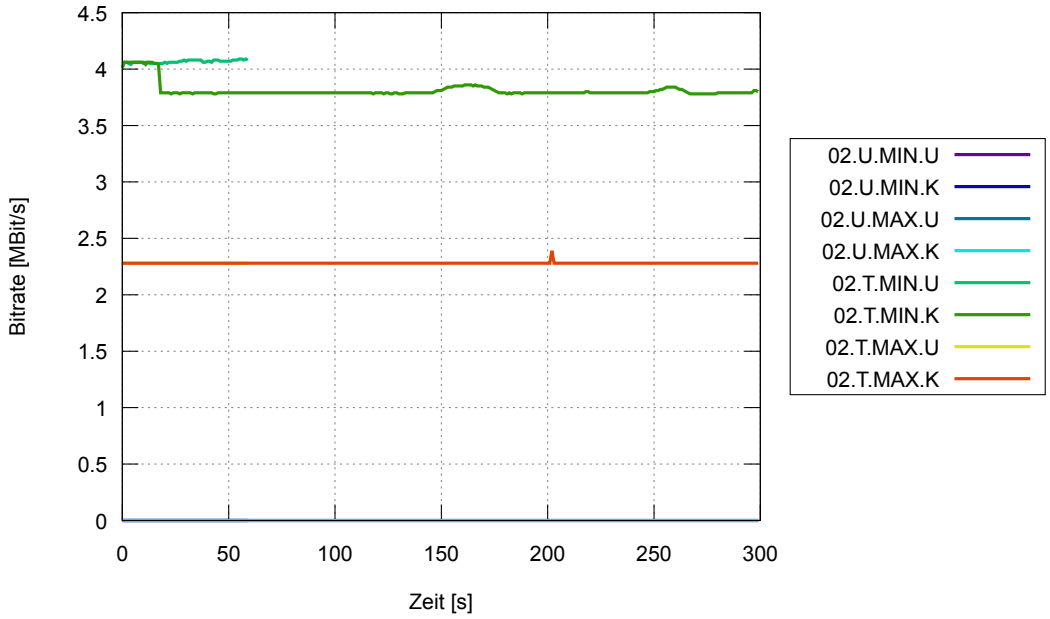


Abbildung 4.12: Resultat Szenario 02 auf shck-Server ohne PRP: TX-Bitrate auf «eth0»-Netzwerk-Interface [MBit/s]

4.2.2 Mit PRP

4.2.2.1 shck-Client

Name	CPU-Last [%]			Systemtime [s]			Usertime [s]		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
02.U.MIN.U	98.99	99.98	100.99	0.12	0.17	0.21	0.78	0.83	0.87
02.U.MIN.K	98.99	99.99	100.99	0.16	0.24	0.32	0.68	0.76	0.84
02.U.MAX.U	35.00	35.98	37.00	0.03	0.06	0.09	0.28	0.30	0.32
02.U.MAX.K	35.00	36.71	38.00	0.03	0.06	0.12	0.25	0.31	0.34
02.T.MIN.U	98.99	99.97	100.99	0.08	0.11	0.16	0.85	0.89	0.92
02.T.MIN.K	98.99	99.99	100.99	0.05	0.11	0.16	0.84	0.89	0.95
02.T.MAX.U	51.99	53.30	54.99	0.08	0.12	0.16	0.37	0.42	0.45
02.T.MAX.K	49.00	52.71	58.99	0.06	0.11	0.16	0.37	0.41	0.48

Tabelle 4.13: Resultat Szenario 02 auf shck-Client mit PRP: CPU-Last, System- / Usertime der shck-Applikation

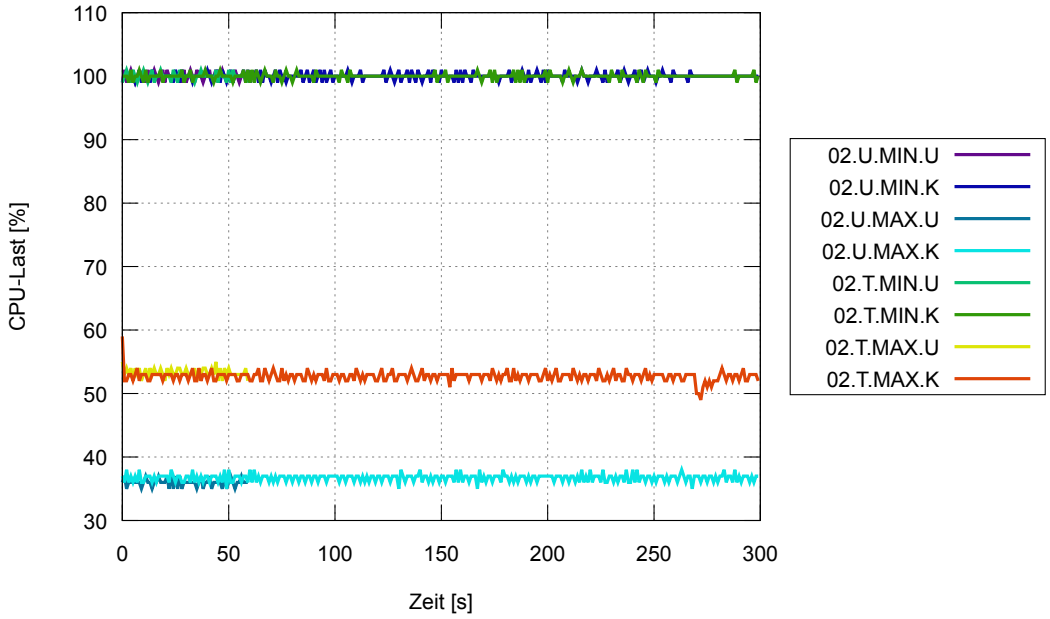


Abbildung 4.13: Resultat Szenario 02 auf shck-Client mit PRP: CPU-Last der shck-Applikation

Name	RX-Bitrate [MBit/s]								
	prp1			eth0			eth1		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
02.T.MIN.U	1.74	1.76	1.76	2.01	2.02	2.03	2.01	2.02	2.03
02.T.MIN.K	1.72	1.77	1.78	1.98	2.04	2.05	1.98	2.04	2.05
02.T.MAX.U	2.15	2.15	2.16	2.48	2.48	2.48	2.48	2.48	2.48
02.T.MAX.K	2.12	2.15	2.25	2.44	2.48	2.59	2.44	2.48	2.59

Tabelle 4.14: Resultat Szenario 02 auf shck-Client mit PRP: RX-Bitrate [MBit/s]

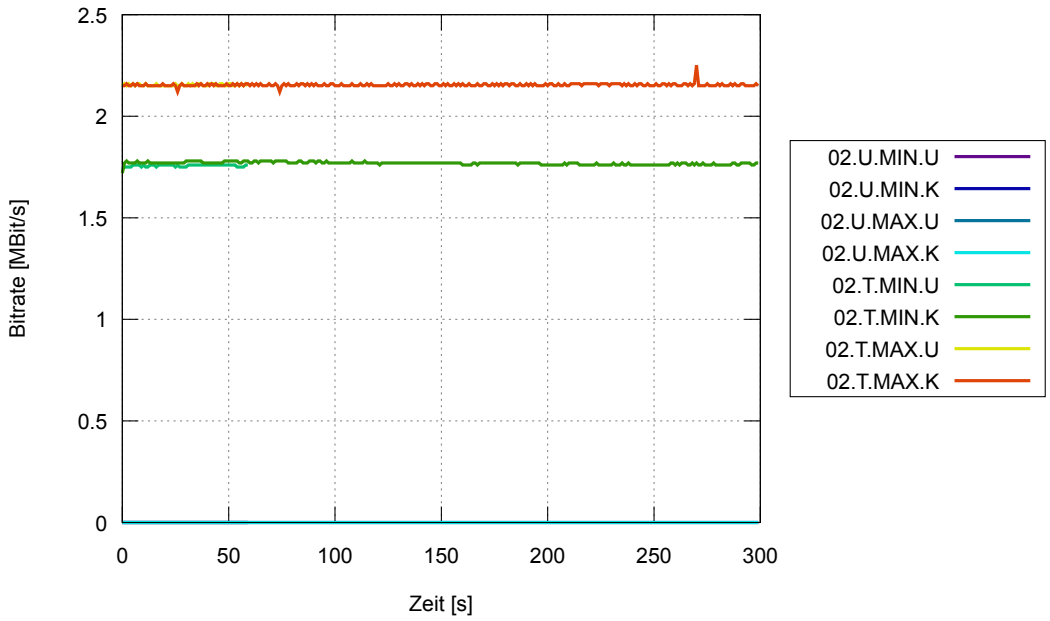


Abbildung 4.14: Resultat Szenario 02 auf shck-Client mit PRP: RX-Bitrate auf «prp1»-Netzwerk-Interface [MBit/s]

Name	TX-Bitrate [MBit/s]								
	prp1			eth0			eth1		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
02.U.MIN.U	7.05	7.19	7.25	8.22	8.39	8.46	8.22	8.39	8.46
02.U.MIN.K	6.83	7.05	7.17	7.97	8.23	8.36	7.97	8.23	8.36
02.U.MAX.U	97.69	98.06	98.47	98.66	98.70	98.72	98.66	98.71	98.72
02.U.MAX.K	97.64	98.06	98.45	98.66	98.71	98.72	98.66	98.71	98.73
02.T.MIN.U	1.92	1.93	1.94	2.18	2.20	2.20	2.18	2.20	2.20
02.T.MIN.K	1.88	1.94	1.95	2.14	2.21	2.22	2.14	2.21	2.22
02.T.MAX.U	98.01	98.05	98.09	98.65	98.70	98.72	98.65	98.70	98.74
02.T.MAX.K	96.39	98.05	99.28	97.03	98.70	98.74	97.03	98.70	98.76

Tabelle 4.15: Resultat Szenario 02 auf shck-Client mit PRP: TX-Bitrate [MBit/s]

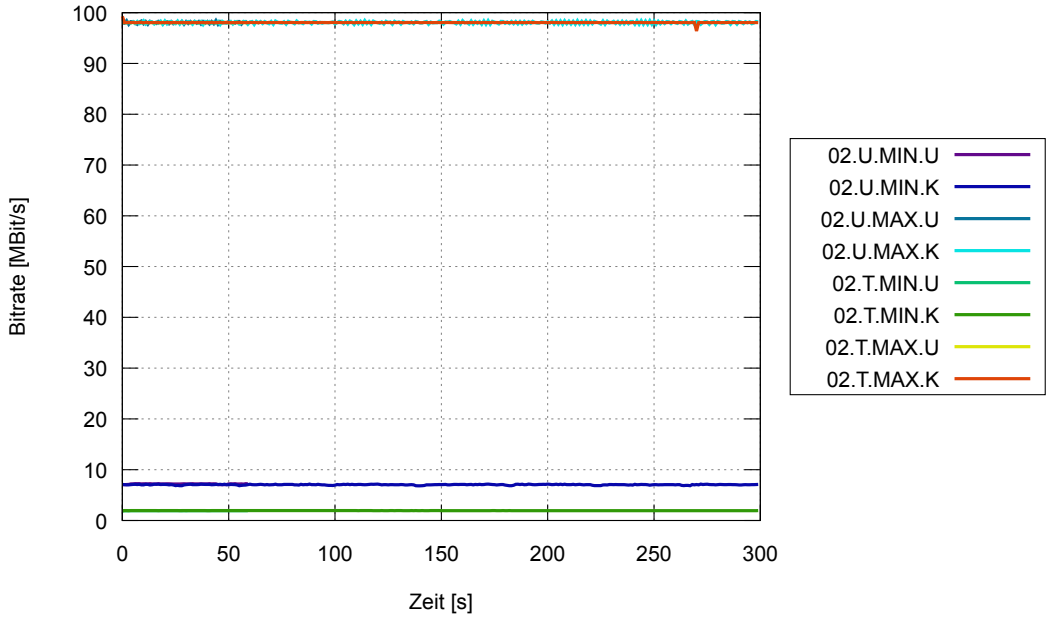


Abbildung 4.15: Resultat Szenario 02 auf shck-Client mit PRP: TX-Bitrate auf «prp1»-Netzwerk-Interface [MBit/s]

4.2.2.2 shck-Server

Name	CPU-Last [%]			Systemtime [s]			Usertime [s]		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
02.U.MIN.U	39.00	57.43	59.00	0.13	0.19	0.28	0.26	0.38	0.45
02.U.MIN.K	52.99	55.84	58.00	0.10	0.18	0.26	0.30	0.38	0.45
02.U.MAX.U	19.00	31.53	38.00	0.08	0.11	0.15	0.09	0.20	0.27
02.U.MAX.K	16.00	31.05	38.00	0.06	0.12	0.17	0.10	0.19	0.27
02.T.MIN.U	19.00	20.80	22.00	0.04	0.09	0.12	0.09	0.12	0.16
02.T.MIN.K	18.00	20.43	22.00	0.04	0.09	0.14	0.06	0.11	0.16
02.T.MAX.U	42.99	46.31	51.00	0.17	0.21	0.25	0.22	0.26	0.31
02.T.MAX.K	42.00	45.51	50.00	0.15	0.20	0.25	0.20	0.25	0.31

Tabelle 4.16: Resultat Szenario 02 auf shck-Server mit PRP: CPU-Last, System- / Usertime der shck-Applikation

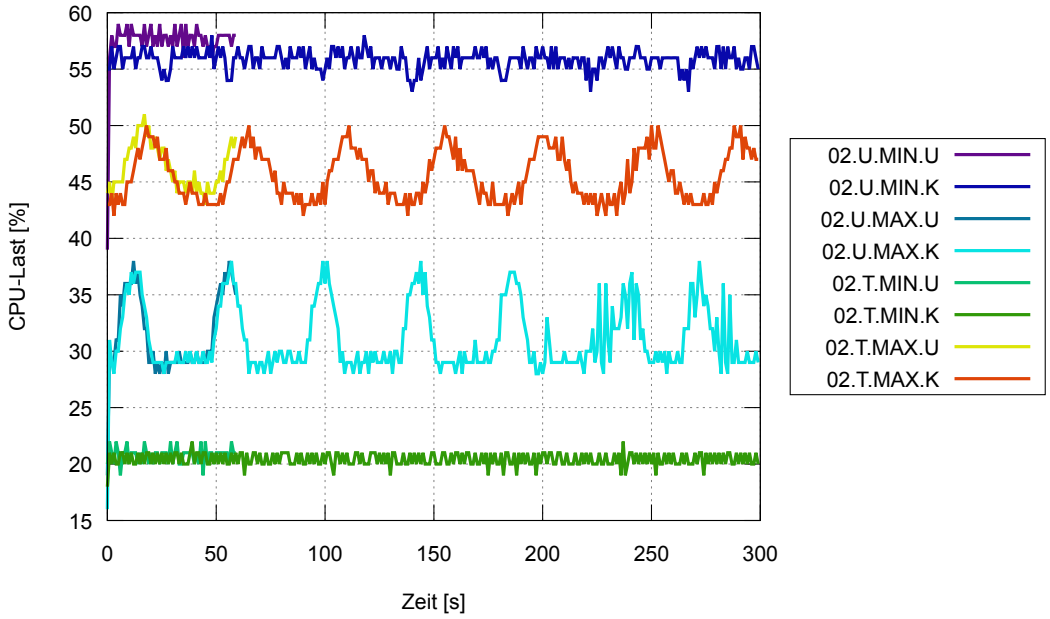


Abbildung 4.16: Resultat Szenario 02 auf shck-Server mit PRP: CPU-Last der shck-Applikation

Name	RX-Bitrate [MBit/s]								
	prp1			eth0			eth1		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
02.U.MIN.U	5.22	7.16	7.26	6.09	8.35	8.47	6.09	8.35	8.47
02.U.MIN.K	6.82	7.05	7.17	7.96	8.23	8.36	7.96	8.23	8.36
02.U.MAX.U	63.72	97.48	98.07	64.14	98.12	98.75	64.09	98.12	98.75
02.U.MAX.K	56.68	97.91	98.07	57.05	98.56	98.75	57.05	98.56	98.75
02.T.MIN.U	1.86	1.93	1.94	2.11	2.19	2.20	2.11	2.19	2.20
02.T.MIN.K	1.63	1.94	1.95	1.86	2.21	2.22	1.86	2.21	2.22
02.T.MAX.U	98.03	98.04	98.07	98.68	98.70	98.75	98.63	98.70	98.75
02.T.MAX.K	97.77	98.04	98.07	98.42	98.69	98.75	98.43	98.69	98.75

Tabelle 4.17: Resultat Szenario 02 auf shck-Server mit PRP: RX-Bitrate [MBit/s]

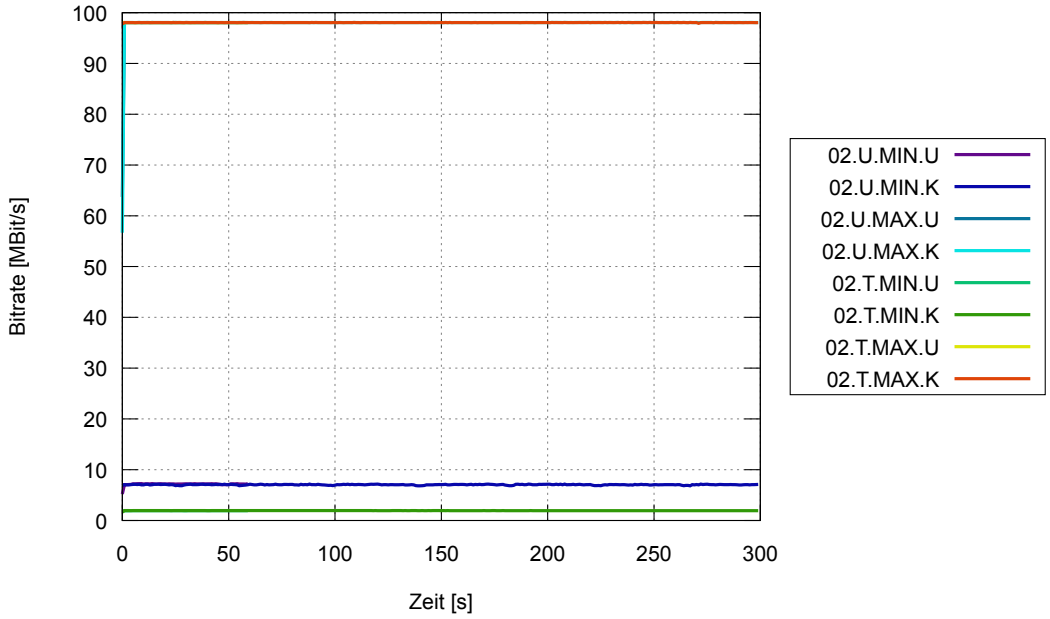


Abbildung 4.17: Resultat Szenario 02 auf shck-Server mit PRP: RX-Bitrate auf «prp1»-Netzwerk-Interface [MBit/s]

Name	**TX-Bitrate [MBit/s]**								
	prp1			**eth0**			**eth1**		
	Min	**Avg**	**Max**	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Max**
02.T.MIN.U	1.70	1.76	1.76	1.95	2.02	2.03	1.95	2.02	2.03
02.T.MIN.K	1.50	1.77	1.78	1.73	2.04	2.05	1.73	2.04	2.05
02.T.MAX.U	2.15	2.15	2.17	2.48	2.48	2.50	2.48	2.48	2.50
02.T.MAX.K	2.12	2.15	2.25	2.44	2.48	2.58	2.44	2.48	2.58

Tabelle 4.18: Resultat Szenario 02 auf shck-Server mit PRP: TX-Bitrate [MBit/s]

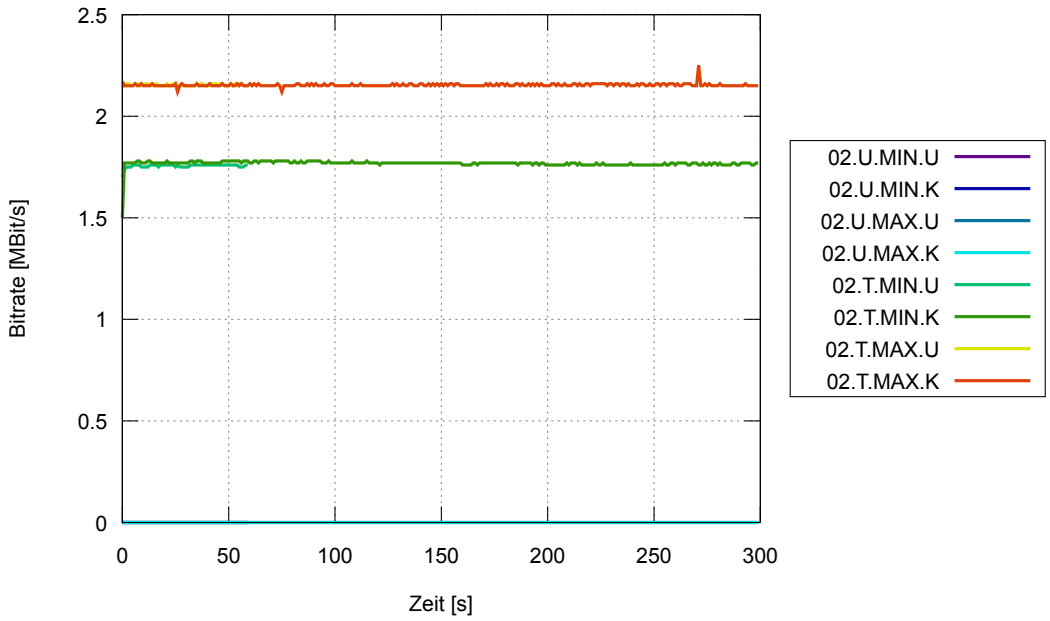


Abbildung 4.18: Resultat Szenario 02 auf shck-Server mit PRP: TX-Bitrate auf «prp1»-Netzwerk-Interface [MBit/s]

4.2.3 Interpretation

4.2.3.1 Limitierungen / Engpässe

4.2.3.2 Verhalten zwischen shck-Client und -Server

4.2.3.3 Verhalten des PRP-1 stacks

4.2.3.4 Weitere Aspekte

4.3 Szenario 03: Effekt von Laufzeitunterscheidungen zwischen Netzwerk A und B

4.3.1 Interpretation

4.3.1.1 Limitierungen / Engpässe

4.3.1.2 Verhalten zwischen shck-Client und -Server

4.3.1.3 Verhalten des PRP-1 stacks

4.3.1.4 Weitere Aspekte

4.4 Szenario 04: Zeitweiser Ausfall eines Netzwerkpfades

Innerhalb dieses Szenarios werden aus Platzgründen lediglich folgende Merkmale pro Netzwerklast und Fall aufgelistet:

- CPU-Last [%] (Durchschnitt)
- Netzwerkbelastung Netzwerk-Interface «eth0»
  - RX-Bitrate (Download) [MBit/s] (Durchschnitt)
  - TX-Bitrate (Download) [MBit/s] (Durchschnitt)

Das Interface «eth0» ist welches, dessen Durchsatz beinträchtigt wird. Die durschnittlichen RX-/TX-Bitraten des «prp1»-Netzwerk-Interfaces können in Szenario 01 in Kapitel 4.1 auf Seite 104 eingesehen werden, da diese vom Ausstieg einer einzelnen NIC nicht beeinflusst werden. Analog dazu gilt das für das aktive Interface «eth1».

4.4.1 Numerische Ergebnisse

4.4.1.1 shck-Client

Fall	Merkmal	U.MIN.U	U.MIN.K	U.MAX.U	U.MAX.K	T.MIN.U	T.MIN.K	T.MAX.U	T.MAX.K
A	CPU [%]	63.63	62.54	16.72	16.66	31.01	30.90	35.51	35.07
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	TX [MBit/s]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	CPU [%]	64.71	63.62	17.38	17.42	31.91	31.77	37.51	36.68
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.73	0.75	1.07	0.94
	TX [MBit/s]	3.23	3.70	39.91	39.66	0.87	0.88	45.48	40.49
8	CPU [%]	63.66	62.61	17.33	17.40	31.90	31.71	36.65	37.91
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.80	0.75	0.96	0.91
	TX [MBit/s]	3.23	3.26	34.97	38.36	0.92	0.88	41.66	39.48
7	CPU [%]	64.20	64.13	17.33	17.40	33.01	32.01	36.15	36.91
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.74	0.68	0.79	0.83
	TX [MBit/s]	3.21	3.36	40.64	37.78	0.88	0.82	34.99	36.29
6	CPU [%]	63.28	63.67	17.42	17.56	32.25	32.08	36.63	36.17
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.62	0.63	0.79	0.78
	TX [MBit/s]	3.07	3.25	36.12	35.90	0.79	0.80	35.97	36.03
5	CPU [%]	62.95	62.64	17.33	17.36	31.58	31.85	36.88	36.11
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.55	0.56	0.69	0.68
	TX [MBit/s]	2.85	2.85	33.15	33.12	0.74	0.73	33.11	33.20
4	CPU [%]	63.78	63.82	17.13	17.17	31.33	31.69	36.48	35.44
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.48	0.47	0.61	0.58
	TX [MBit/s]	2.30	2.49	27.57	29.11	0.68	0.66	31.41	29.58
3	CPU [%]	62.98	62.82	17.10	17.16	31.68	31.67	36.30	35.70
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.28	0.28	0.37	0.37
	TX [MBit/s]	1.93	1.95	21.64	21.54	0.48	0.48	21.81	21.66
2	CPU [%]	59.93	62.99	17.10	17.06	31.91	31.49	35.93	35.07
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	TX [MBit/s]	2.04	2.24	24.28	24.27	0.54	0.54	24.46	24.27
1	CPU [%]	60.46	63.38	16.67	16.66	30.76	31.09	34.13	34.97
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	TX [MBit/s]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabelle 4.19: Resultat Szenario 04 auf shck-Client: CPU-Last des PRP-1 stacks, RX-/TX-Bitrate auf «eth0»-Netzwerk-Interface [MBit/s]

4.4.1.2 shck-Server

Fall	Merkmal	U.MIN.U	U.MIN.K	U.MAX.U	U.MAX.K	T.MIN.U	T.MIN.K	T.MAX.U	T.MAX.K
A	CPU [%]	54.71	59.57	29.86	29.82	26.65	26.67	50.40	50.76
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	TX [MBit/s]	0.00	0.00	0.00	0.00	2.07	2.07	2.48	2.48
9	CPU [%]	65.59	67.26	33.73	35.18	26.98	27.09	55.38	55.00
	RX [MBit/s]	2.97	3.38	33.94	36.66	0.79	0.82	43.17	36.94
	TX [MBit/s]	0.00	0.00	0.00	0.00	2.04	2.04	2.48	2.48
8	CPU [%]	62.14	63.30	33.60	35.08	27.35	27.13	54.51	55.02
	RX [MBit/s]	3.07	2.96	33.95	34.63	0.86	0.82	38.05	36.05
	TX [MBit/s]	0.00	0.00	0.00	0.00	2.05	2.05	2.48	2.48
7	CPU [%]	68.91	67.02	35.03	34.04	26.71	26.96	54.28	54.61
	RX [MBit/s]	2.99	3.14	38.90	34.59	0.80	0.74	31.23	32.81
	TX [MBit/s]	0.00	0.00	0.00	0.00	2.01	2.03	2.48	2.48
6	CPU [%]	62.73	67.33	33.00	33.91	26.88	26.92	53.91	54.53
	RX [MBit/s]	2.58	2.82	29.62	30.70	0.67	0.69	31.19	30.46
	TX [MBit/s]	0.00	0.00	0.00	0.00	2.03	2.04	2.49	2.48
5	CPU [%]	62.91	62.86	32.81	33.79	26.91	26.79	53.90	53.78
	RX [MBit/s]	2.31	2.31	26.83	26.98	0.60	0.61	26.95	26.80
	TX [MBit/s]	0.00	0.00	0.00	0.00	2.05	2.04	2.49	2.48
4	CPU [%]	57.01	62.15	32.58	32.66	27.08	26.74	53.31	53.37
	RX [MBit/s]	1.80	1.94	21.62	22.56	0.52	0.51	23.44	22.73
	TX [MBit/s]	0.00	0.00	0.00	0.00	2.06	2.04	2.49	2.49
3	CPU [%]	62.86	64.70	31.15	31.48	26.65	26.52	51.95	52.08
	RX [MBit/s]	1.20	1.12	12.07	13.10	0.30	0.31	13.66	13.89
	TX [MBit/s]	0.00	0.00	0.00	0.00	2.04	2.04	2.49	2.49
2	CPU [%]	59.73	62.69	29.58	29.65	26.30	26.36	50.05	50.35
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	TX [MBit/s]	0.00	0.00	0.00	0.00	2.04	2.05	2.47	2.47
1	CPU [%]	60.18	62.94	29.78	29.77	26.70	26.57	50.23	50.57
	RX [MBit/s]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	TX [MBit/s]	0.00	0.00	0.00	0.00	2.06	2.06	2.48	2.48

Tabelle 4.20: Resultat Szenario 04 auf shck-Server: CPU-Last des PRP-1 stacks, RX-/TX-Bitrate auf «eth0»-Netzwerk-Interface [MBit/s]

4.4.2 Graphische Resultate

Es erfolgen zu jeder generierten Netzwerklast 6 Abbildungen (3 Merkmale (siehe Beginn dieses Kapitels) pro Teilnehmer), welche je alle 10 Fälle abdecken.

Die in Kapitel 3.3.4.6 auf Seite 101 beschriebene Fall-Bezeichnung ist bei der Bezeichnung der entsprechenden Netzwerklast angehängt und weist auf den verwendeten Intervall in Sekunden oder den Komplettausfall hin.

U.MIN.U

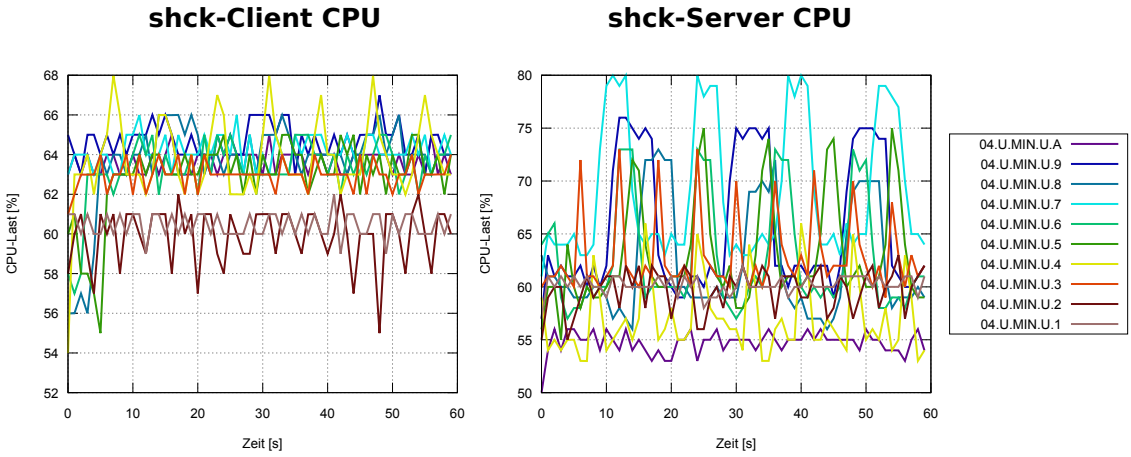


Abbildung 4.19: Resultat Szenario 04: CPU-Last des PRP-1 stacks der Netzwerklast U.MIN.U auf shck-Client und -Server

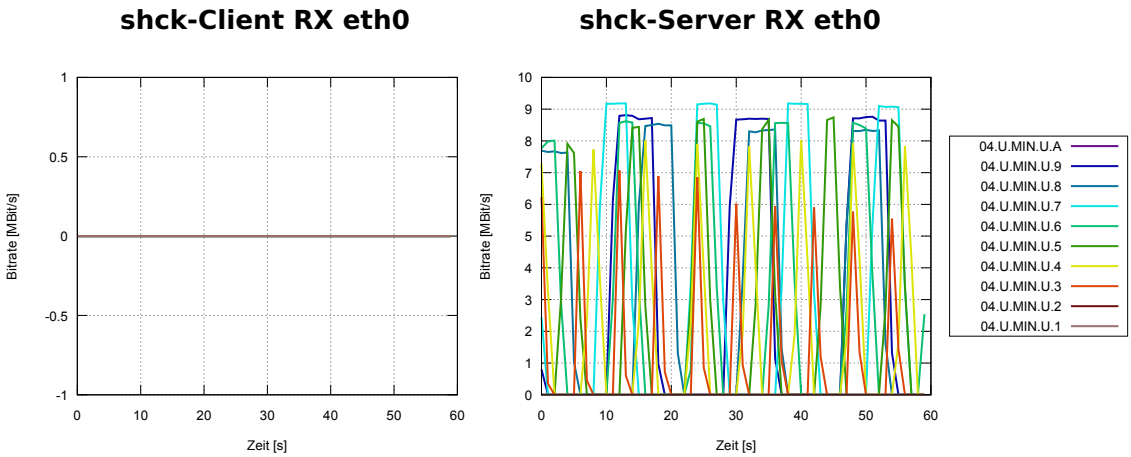


Abbildung 4.20: Resultat Szenario 04: RX-Bitrate auf Interface «eth0» der Netzwerklast U.MIN.U auf shck-Client und -Server

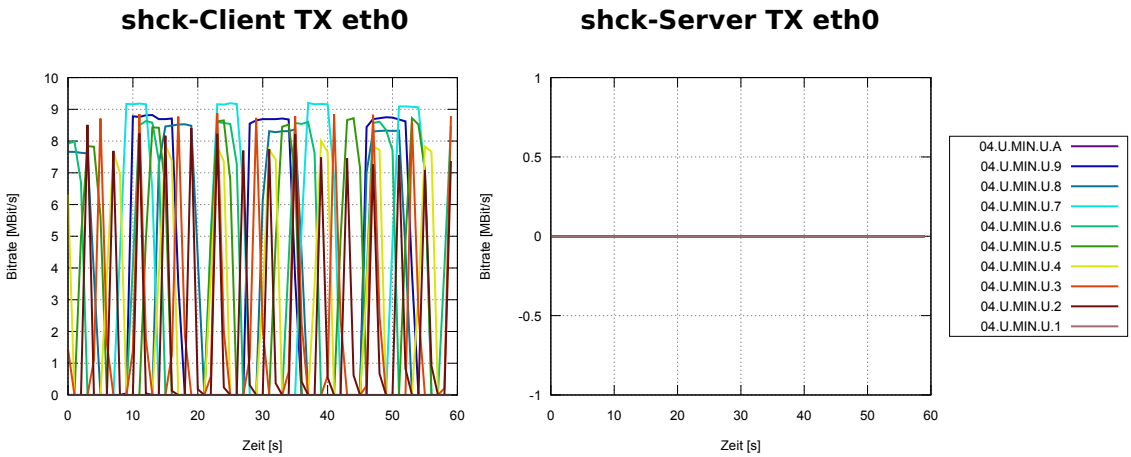


Abbildung 4.21: Resultat Szenario 04: TX-Bitrate auf Interface «eth0» der Netzwerklast U.MIN.U auf shck-Client und -Server

U.MIN.K

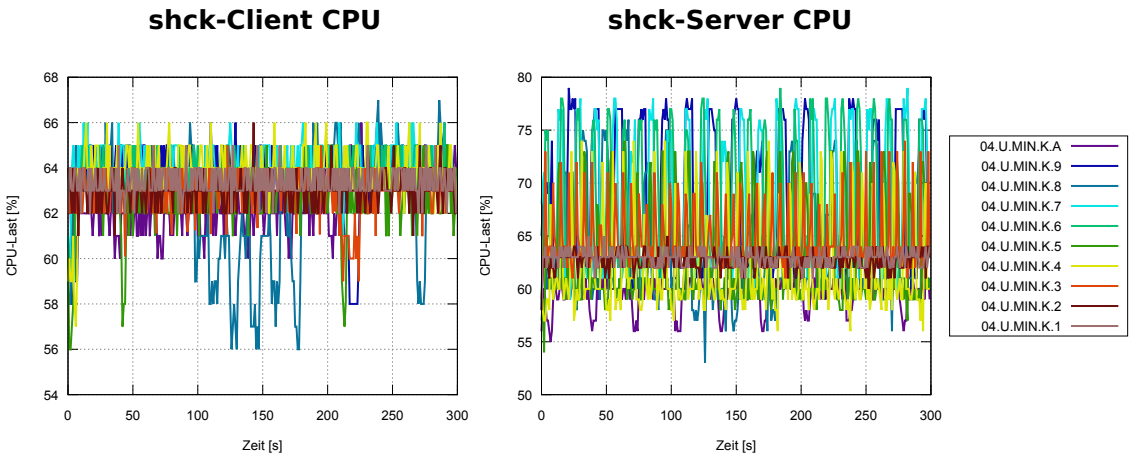


Abbildung 4.22: Resultat Szenario 04: CPU-Last des PRP-1 stacks der Netzwerklast U.MIN.K auf shck-Client und -Server

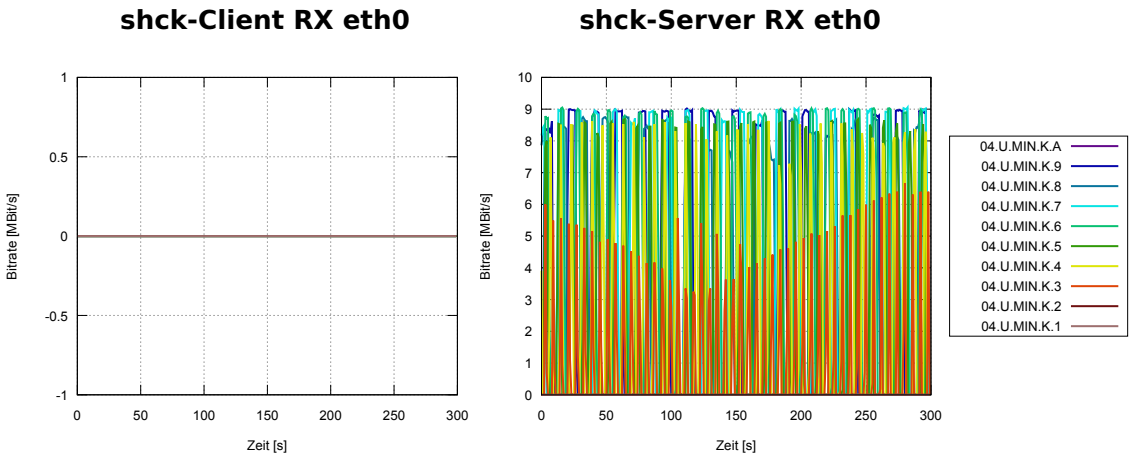


Abbildung 4.23: Resultat Szenario 04: RX-Bitrate auf Interface «eth0» der Netzwerklast U.MIN.K auf shck-Client und -Server

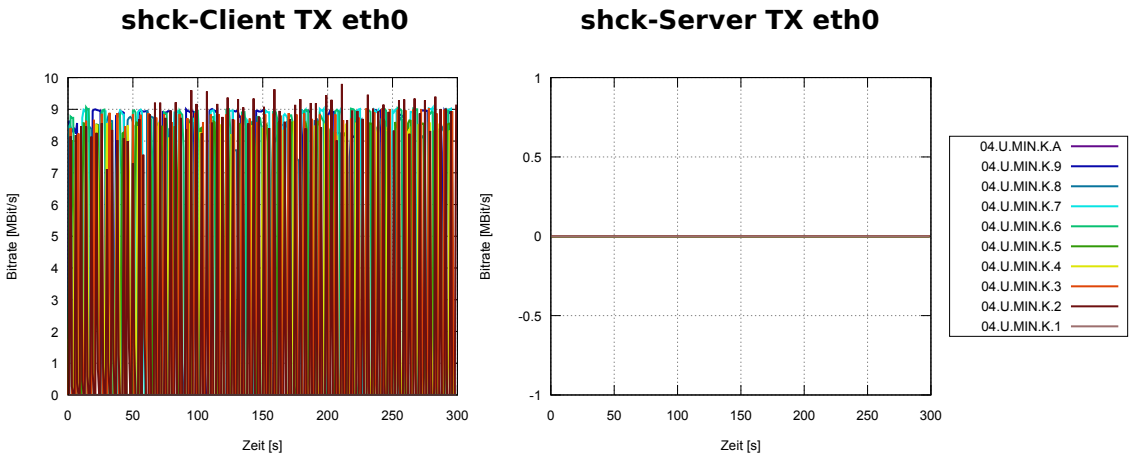


Abbildung 4.24: Resultat Szenario 04: TX-Bitrate auf Interface «eth0» der Netzwerklast U.MIN.K auf shck-Client und -Server

U.MAX.U

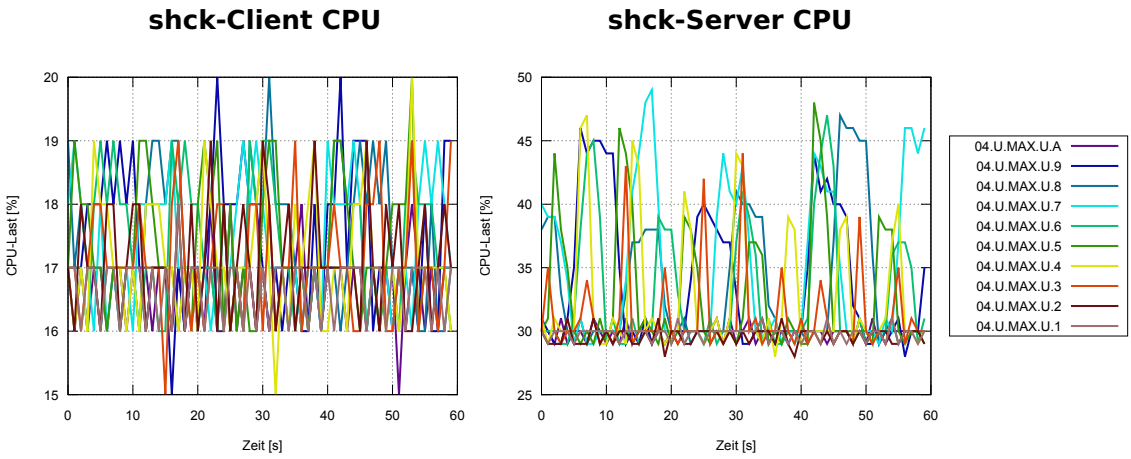


Abbildung 4.25: Resultat Szenario 04: CPU-Last des PRP-1 stacks der Netzwerklast U.MAX.U auf shck-Client und -Server

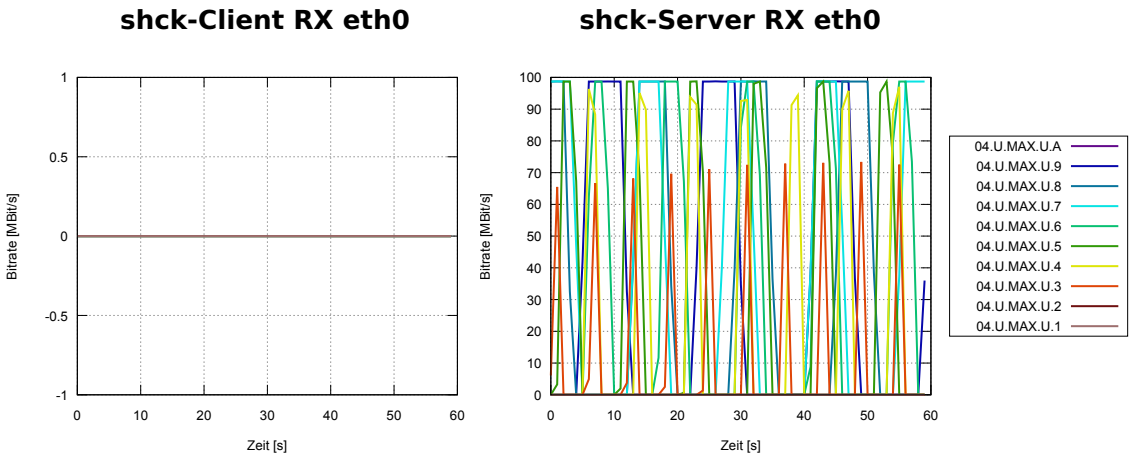


Abbildung 4.26: Resultat Szenario 04: RX-Bitrate auf Interface «eth0» der Netzwerklast U.MAX.U auf shck-Client und -Server

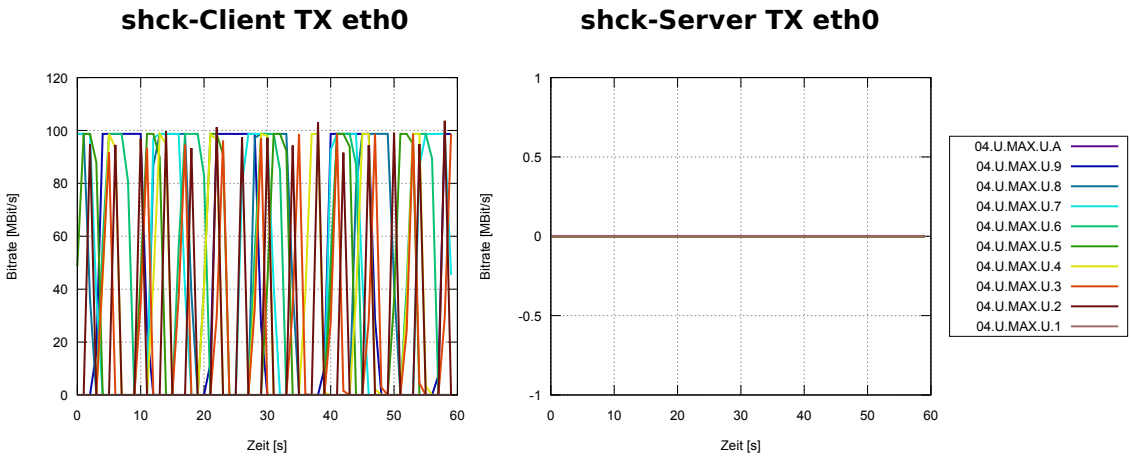


Abbildung 4.27: Resultat Szenario 04: TX-Bitrate auf Interface «eth0» der Netzwerklast U.MAX.U auf shck-Client und -Server

U.MAX.K

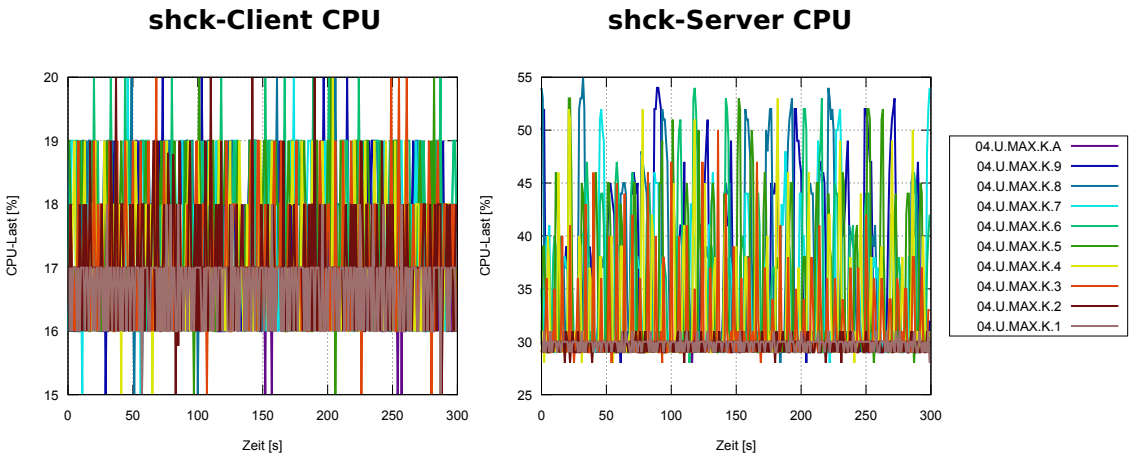


Abbildung 4.28: Resultat Szenario 04: CPU-Last des PRP-1 stacks der Netzwerklast U.MAX.K auf shck-Client und -Server

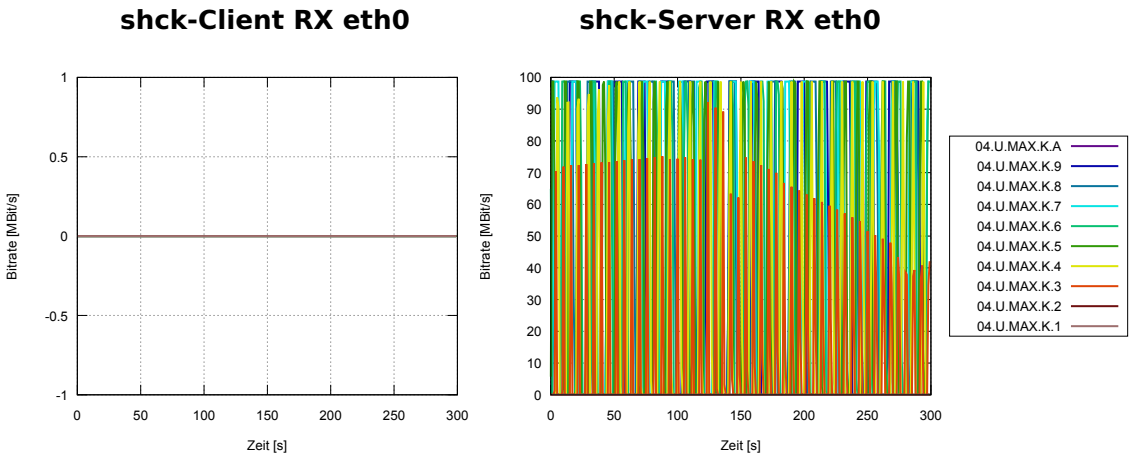


Abbildung 4.29: Resultat Szenario 04: RX-Bitrate auf Interface «eth0» der Netzwerklast U.MAX.K auf shck-Client und -Server

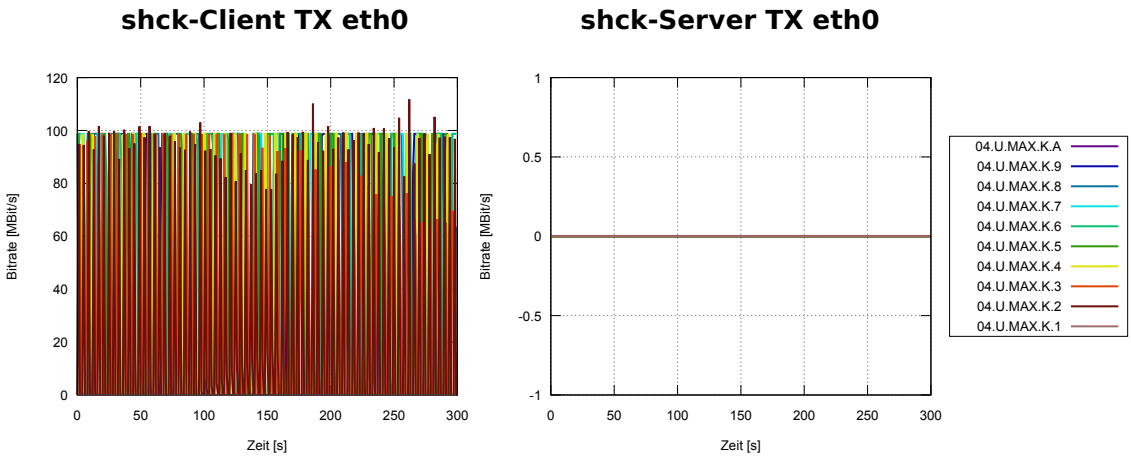


Abbildung 4.30: Resultat Szenario 04: TX-Bitrate auf Interface «eth0» der Netzwerklast U.MAX.K auf shck-Client und -Server

T.MIN.U

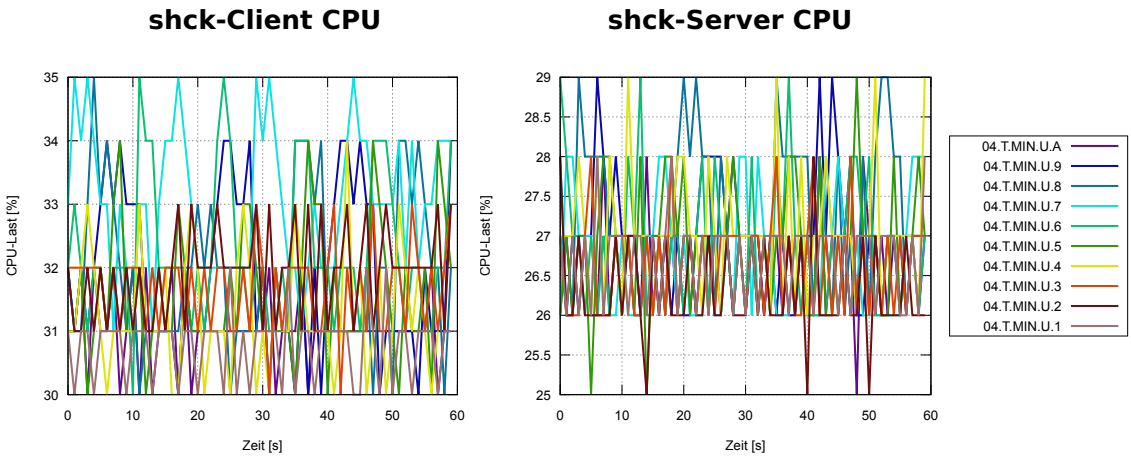


Abbildung 4.31: Resultat Szenario 04: CPU-Last des PRP-1 stacks der Netzwerklast T.MIN.U auf shck-Client und -Server

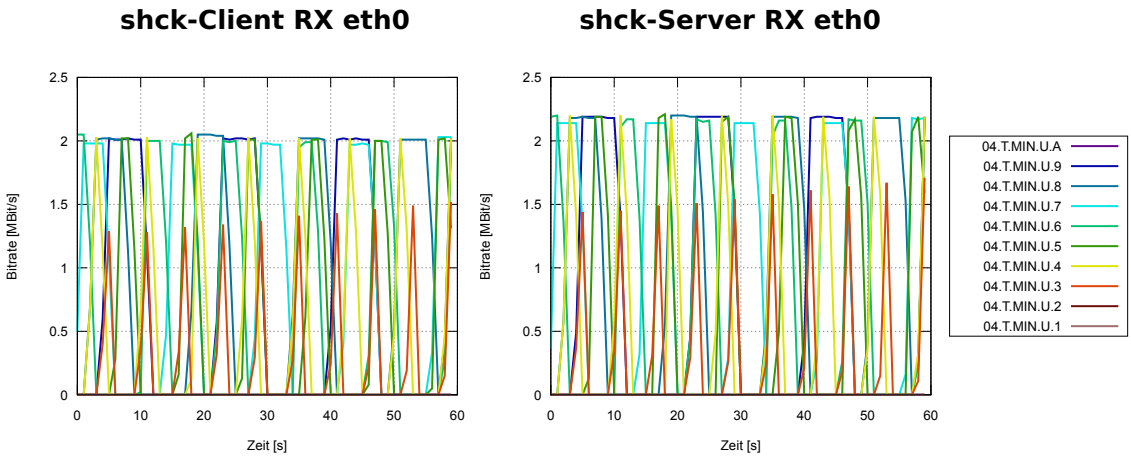


Abbildung 4.32: Resultat Szenario 04: RX-Bitrate auf Interface «eth0» der Netzwerklast T.MIN.U auf shck-Client und -Server

Abbildung 4.33: Resultat Szenario 04: TX-Bitrate auf Interface «eth0» der Netzwerklast T.MIN.U auf shck-Client und -Server

T.MIN.K

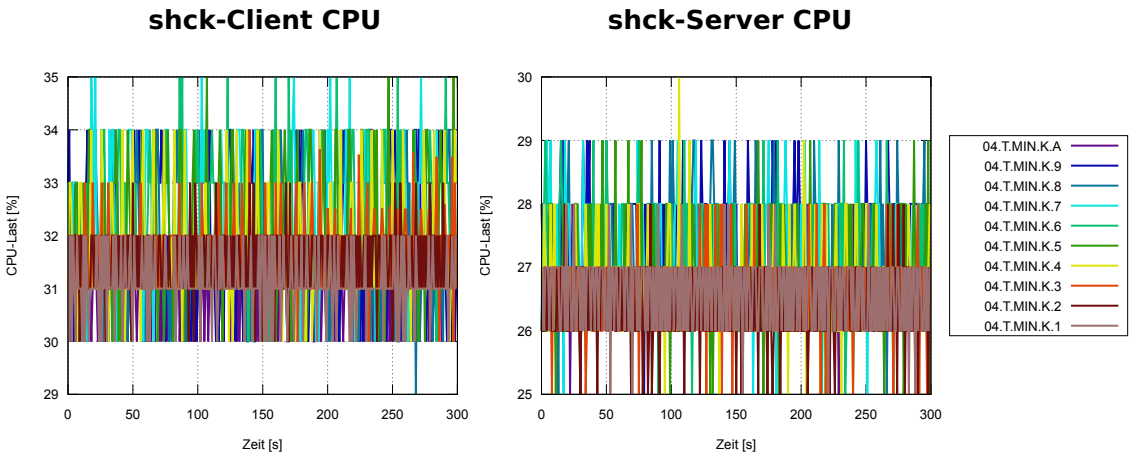


Abbildung 4.34: Resultat Szenario 04: CPU-Last des PRP-1 stacks der Netzwerklast T.MIN.K auf shck-Client und -Server

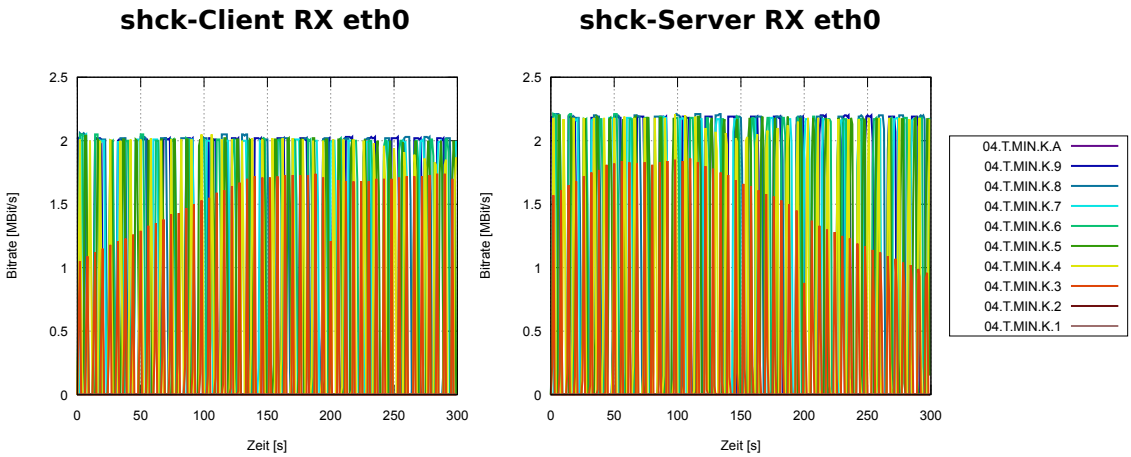


Abbildung 4.35: Resultat Szenario 04: RX-Bitrate auf Interface «eth0» der Netzwerklast T.MIN.K auf shck-Client und -Server

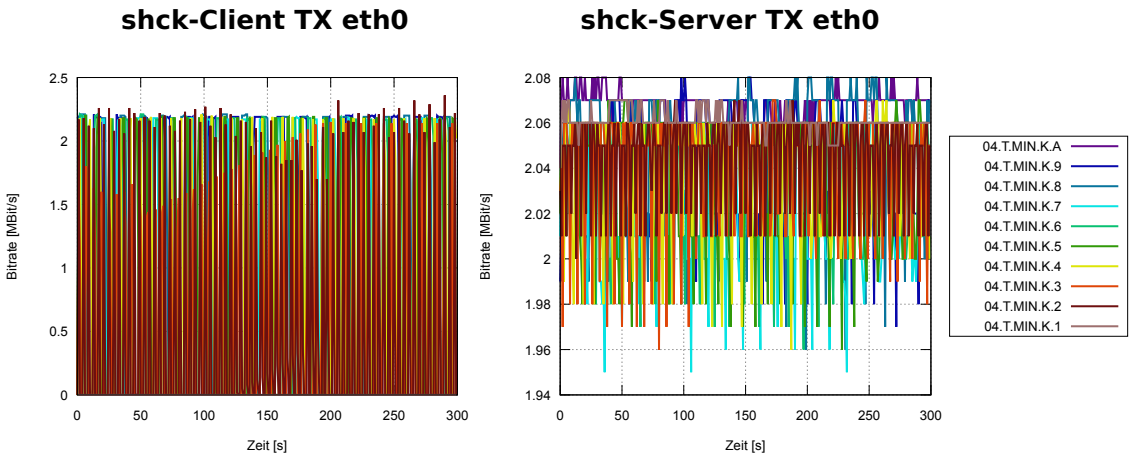


Abbildung 4.36: Resultat Szenario 04: TX-Bitrate auf Interface «eth0» der Netzwerklast T.MIN.K auf shck-Client und -Server

T.MAX.U

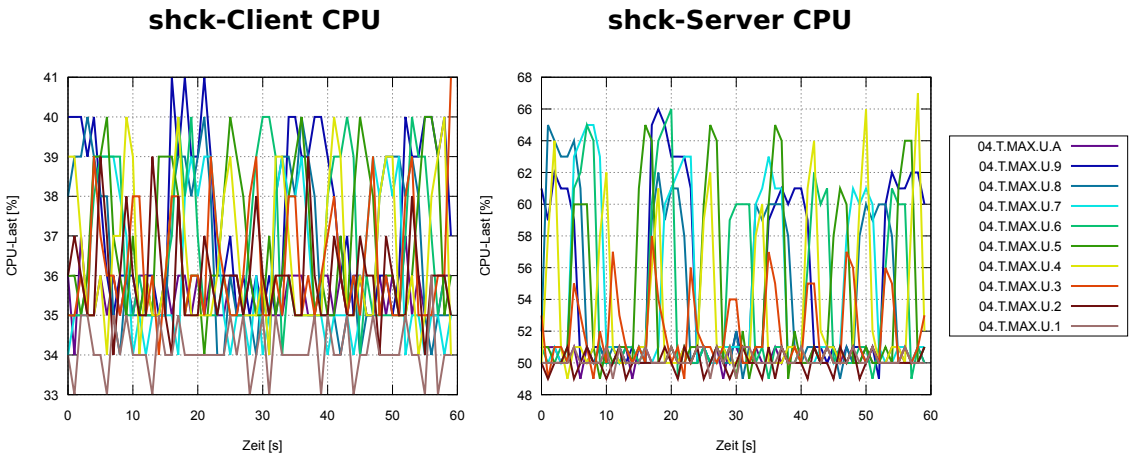


Abbildung 4.37: Resultat Szenario 04: CPU-Last des PRP-1 stacks der Netzwerklast T.MAX.U auf shck-Client und -Server

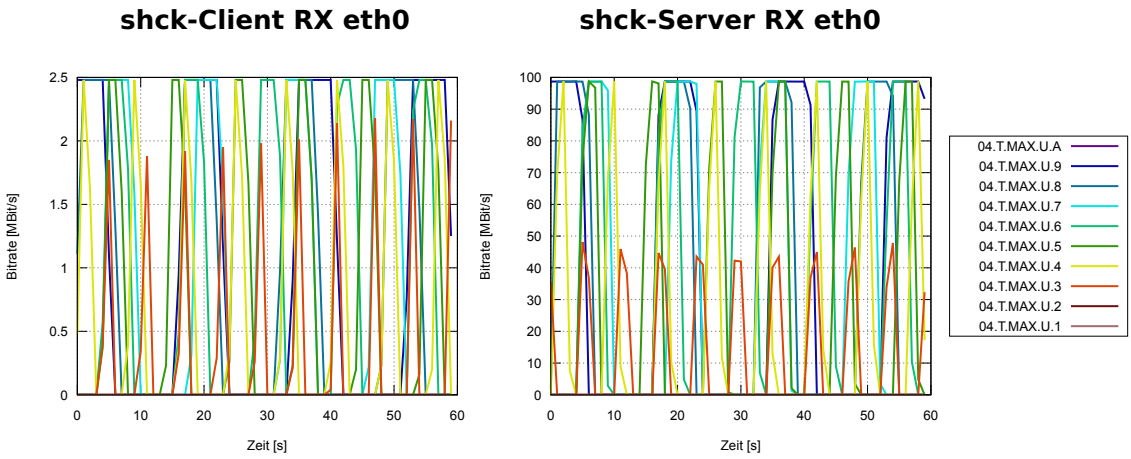


Abbildung 4.38: Resultat Szenario 04: RX-Bitrate auf Interface «eth0» der Netzwerklast T.MAX.U auf shck-Client und -Server

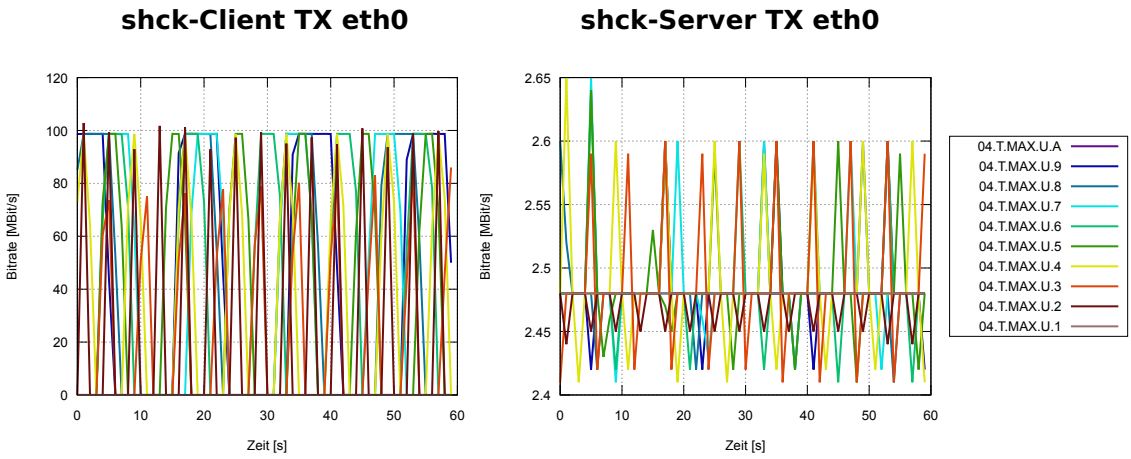


Abbildung 4.39: Resultat Szenario 04: TX-Bitrate auf Interface «eth0» der Netzwerklast T.MAX.U auf shck-Client und -Server

T.MAX.K

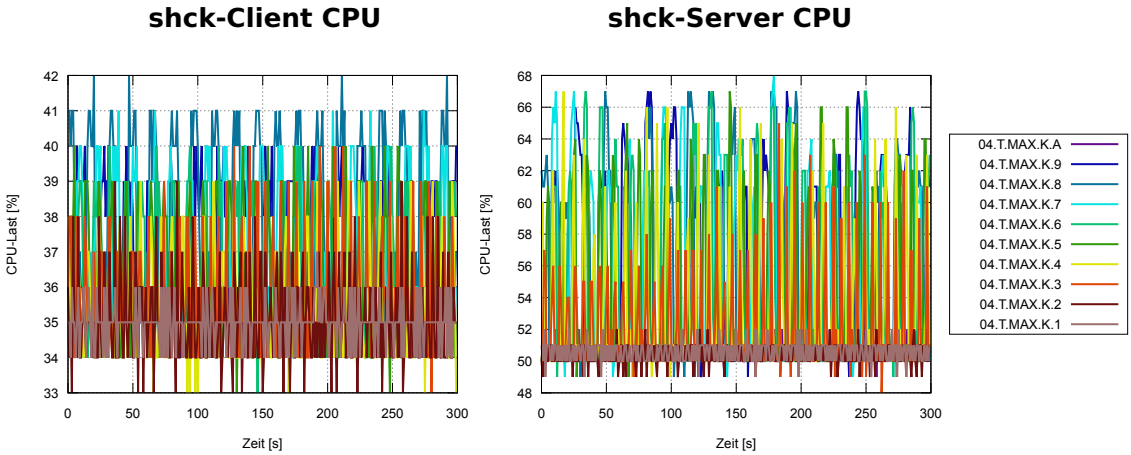


Abbildung 4.40: Resultat Szenario 04: CPU-Last des PRP-1 stacks der Netzwerklast T.MAX.K auf shck-Client und -Server

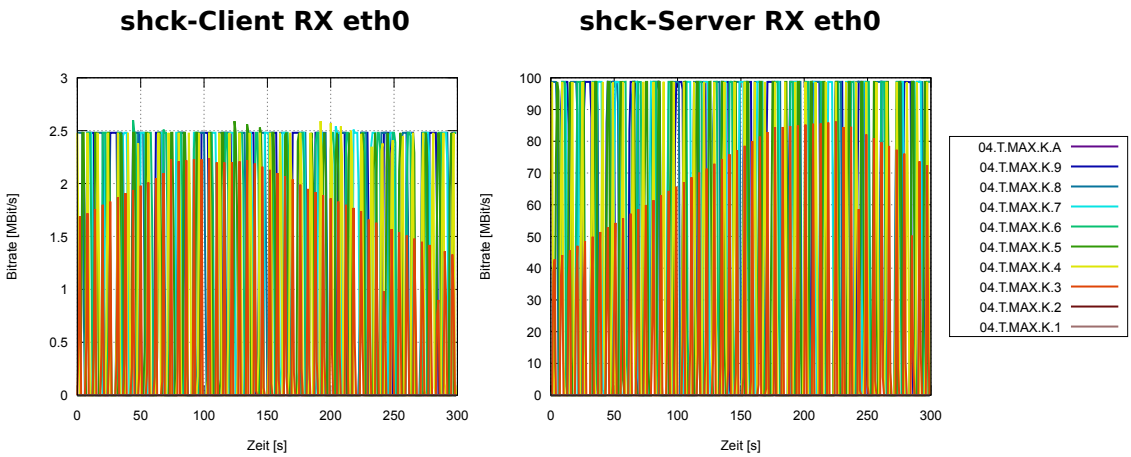


Abbildung 4.41: Resultat Szenario 04: RX-Bitrate auf Interface «eth0» der Netzwerklast T.MAX.K auf shck-Client und -Server

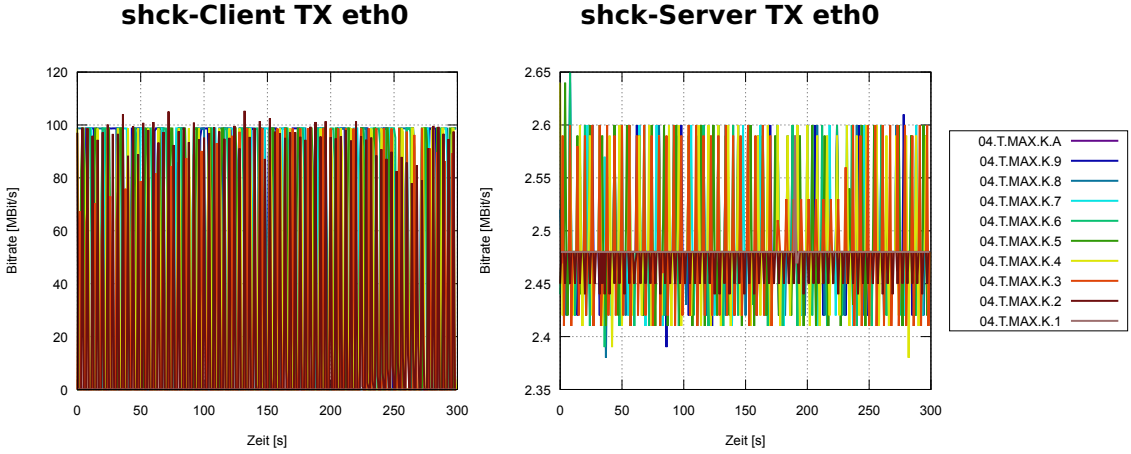


Abbildung 4.42: Resultat Szenario 04: TX-Bitrate auf Interface «eth0» der Netzwerklast T.MAX.K auf shck-Client und -Server

4.4.3 Interpretation

4.4.3.1 Limitierungen / Engpässe

4.4.3.2 Verhalten zwischen shck-Client und -Server

4.4.3.3 Verhalten des PRP-1 stacks

4.4.3.4 Weitere Aspekte

- «prp1» und «eth1» Bitraten werden nicht beeinflusst / keine Limitierung -> nur «eth0»
- shck-Client
  - Kein Einfluss auf CPU bei keinem Fall
  - 1s kein Durchfluss -> Zeit für Deaktivieren/Aktivieren ECI-Box zu lange so-dass eth0 nicht wirklich aktiv wird
  - 2s kleine TX-Bitraten jedoch bei TCP keine/kaum Quittierungen vom Server
  - 3s kleine Bitraten
  - 4s etwas bessere Bitraten, danach nur noch marginale Verbesserungen pro erhöhte Sekunde der Intervallgrösse
    - Diese Steigerung pro Intervallgrösse spiegelt sich in den graphischen Resultaten wieder (besonders ersichtlich bei Intervallgrösse von 3s)
- shck-Server
  - Einfluss auf CPU, wird geringer je weniger über «eth0» versendet wird -> Muss weniger Daten verarbeiten
    - Kann so weit kommen, dass Server weniger CPU für selbe Last benötigt als Client (Client sendet über eth0 und eth1, Server erhält nur eth1) (Normalfall benötigt Server mehr als Client, siehe Szenario 01)
  - TX-Bitrate Server entspricht der aus Szenario 01, da Verbindung zwischen «eth0» von «srv02» zum Switch aktiv ist -> Server merkt nichts von Ausfall
- Man sieht in den Grafiken wie die shck-Server TX-bitrate mit der RX-bitrate des Servers korrespondiert. TCP: RX-Bitrate Client erst ab ca. 4s ähnlich zu TX-Bitrate Server weil Server beim Senden nichts vom Ausfall mitbekommt (Client erhält weniger als Server versendet)

- In grafiken bei MAX mit intervall 2s peaks über 100 -> aktivieren interface -> annahme: tx-buffer leeren (einfluss auf statistik-aktualisierung)
- Limitierung liegt in der Zeit, in der über «eth0» effektiv versendet wird -> Weniger versandt, kleinere Bitrate, weniger Server-CPU-Last
- Laut standard PRP-Supervision Frames nur wenn beide Interfaces aktiv, dann erkennt DAN ob anderer Host SAN oder DAN ist. Im PRP-1 stack ist adapter_active immer auf 1, auch wenn Kabel getrennt ist. Der stack sendet sobald adapter_active==1 ist, das heisst er weiss nichts vom Ausfall. Die an das deaktivierte Interface gesendeten Daten sind für den PRP-1 stack gesendet, werden jedoch nicht effektiv vom OS versendet -> korrekte /proc-Statistiken.

4.5 Szenario 05: Abhängigkeit vom verwendeten Protokoll (TCP/UDP)

In diesem Abschnitt wird ein Vergleich zwischen TCP- und UDP-Verkehr vorgenommen. Dazu dienen die Resultate aus Szenario 01 (siehe Kapitel 4.1 auf Seite 104). Der Vergleich zwischen reiner Layer-2- und UDP-Netzwerklast wird in Kapitel 3.3.4.1 auf Seite 90 durchgeführt.

4.5.1 Interpretation

4.5.1.1 Limitierungen / Engpässe

4.5.1.2 Verhalten zwischen shck-Client und -Server

4.5.1.3 Verhalten des PRP-1 stacks

4.5.1.4 Weitere Aspekte

4.6 Szenario 06: Einfluss nicht entfernter Duplikate auf Funktion und Performance

4.6.1 Interpretation

4.6.1.1 Limitierungen / Engpässe

4.6.1.2 Verhalten zwischen shck-Client und -Server

4.6.1.3 Verhalten des PRP-1 stacks

4.6.1.4 Weitere Aspekte

4.7 Szenario 07: Auswirkungen, wenn Frames out-of-sequence ankommen

4.7.1 Interpretation

4.7.1.1 Limitierungen / Engpässe

4.7.1.2 Verhalten zwischen shck-Client und -Server

4.7.1.3 Verhalten des PRP-1 stacks

4.7.1.4 Weitere Aspekte

4.8 Szenario 08: Einluss von Offload-Mechanismen

4.8.1 Interpretation

4.8.1.1 Limitierungen / Engpässe

4.8.1.2 Verhalten zwischen shck-Client und -Server

4.8.1.3 Verhalten des PRP-1 stacks

4.8.1.4 Weitere Aspekte

5 Diskussion und Ausblick

5.1 Besprechung der Ergebnisse

Lorem ipsum

5.2 Erfüllung der Aufgabenstellung

Soll
Ist
Nachweis

Tabelle 5.1: Nachweis:

5.3 Rückblick

Lorem ipsum

5.4 Ausblick

Lorem ipsum

Teil III

Verzeichnisse

6 Literaturverzeichnis

[1] A. Brouwer et al.: *Linux Programmer's Manual, fprintf, @http://man7.org/linux/man-pages/man3/fprintf.3.html*, April 2015.

[2] A. Cahalan: *pmap(1) - Linux man page @http://linux.die.net/man/1/pmap*, Februar 2015.

[3] A. Cahalan: *top(1) - Linux man page @http://linux.die.net/man/1/top*, März 2015.

[4] A. Darby et al.: *Experience using PRP Ethernet redundancy for Substation Automation Systems @http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6822815&sortType%3Dasc_p_Sequence%26filter%3DAND%28p_IS_Number%3A6809516%29*, März 2014.

[5] A. Engelen: *NetHogs @http://nethogs.sourceforge.net*, Februar 2015.

[6] A. Gemperli: *Bedienungsanleitung Impairment Generator (ECI-IG)*, März 2008.

[7] A. Reprakash: *Understanding TCP Window Size / Window Scaling @http://ithitman.blogspot.ch/2013/02/understanding-tcp-window-window-scaling.html*, Februar 2013.

[8] A. Zimmerman et al.: *flowgrind @http://www.flowgrind.net*, Februar 2015.

[9] B. Greer: *Ethernet for Control - Understanding the Basics @http://www.industrialethernetu.com/courses/101_1.htm*, April 2015.

[10] Calnex Solutions Ltd: *Calnex Paragon: User Guide*, Dezember 2009.

[11] D. Eckhardt et al.: *Linux Programmer's Manual, close, @http://man7.org/linux/man-pages/man2/close.2.html*, Dezember 2013.

[12] D. Eckhardt et al.: *Linux Programmer's Manual, read, @http://man7.org/linux/man-pages/man2/read.2.html*, Mai 2014.

[13] D. Miller et al.: *ethtool(8) - Linux man page @http://linux.die.net/man/8/ethtool*, März 2015.

[14] ESnet / Lawrence Berkeley National Laboratory: *iperf3 @http://software.es.net/iperf*, Februar 2015.

[15] Free Software Foundation, Inc.: *Fast Scatter-Gather I/O @http://www.gnu.org/software/libc/manual/html_node/Scatter_002dGather.html*, März 2015.

[16] Future Technology Devices International Ltd.: *Virtual COM Port Drivers @http://www.ftdichip.com/Drivers/VCP.htm*, März 2015.

[17] G. Combs: *tshark - Dump and analyze network traffic @https://www.wireshark.org/docs/man-pages/tshark.html*, Februar 2015.

[18] G. Combs et al.: *Offloading @https://wiki.wireshark.org/CaptureSetup/Offloading*, März 2015.

[19] G. Langeveld: *atop @http://www.atoptool.nl/index.php*, Februar 2015.

[20] H. Gräbner et al.: *Paketumlaufzeit @http://de.wikipedia.org/wiki/Paketumlaufzeit*, Mai 2015.

[21] H. Ware: *vmstat(8) - Linux man page @http://linux.die.net/man/8/vmstat*, März 2015.

[22] H. Weibel: *BA15_wlan_1: Projektarbeit im Fachgebiet Kommunikation*. Aufgabenstellung, Februar 2015.

[23] H. Weibel: *@http://engineering.zhaw.ch/fileadmin/user_upload/engineering/_Institute_und_Zentren/INES/PRP/PRP_Tutorial.pdf*, Februar 2015.

[24] H. Weibel, F. Reichert: *Ethernet Redundancy with zero Switchover Time @www.swisstmeeting.ch/tl_files/images/Communication%20Conference/Unterbrechungsfreie_Redundanz_slides_ZHAW_Reichert.pdf*, Februar 2015.

[25] Hirschmann GmbH: *PRP - Parallel Redundancy Protocol @http://www.hirschmann.com/en/Hirschmann_Produkte/Industrial_Ethernet/Technologies/PRP_-_Parallel_Redundancy_Protocol/index.phtml*, Februar 2015.

[26] Institute of Embedded Systems: *ECI-1588 @http://www.zhaw.ch/de/engineering/institute-zentren/ines/produkte-und-dienstleistungen/ptp-ieee-1588/ptp-cable-interceptor.html*, März 2015.

[27] Institute of Embedded Systems: *PRP-1 Software Stack @http://ines.zhaw.ch/de/engineering/institute-zentren/ines/produkte-und-dienstleistungen/high-availability/prp-1-software-stack.html*, Februar 2015.

[28] Institute of Embedded Systems: *PRP @http://ines.zhaw.ch/de/engineering/institute-zentren/ines/forschung-und-entwicklung/praezise-zeitsynchronisation-und-hochverfuegbare-netze/technologien/prp-technologie.html*, Februar 2015.

[29] J. Corbet: *Generic recieve offload @https://lwn.net/Articles/358910/*, Oktober 2009.

[30] J. Damato: *strace: for fun, profit, and debugging @http://timetobleed.com/hello-world/*, Februar 2015.

[31] J. Gross: *vlan: Centralize handling of hardware acceleration @http://permalink.gmane.org/gmane.linux.network/175502*, März 2015.

[32] J. Henderson: *What is RSS and VSZ in Linux memory management @http://stackoverflow.com/a/21049737*, März 2015.

[33] J. Seward et al.: *Valgrind @http://valgrind.org/*, März 2015.

[34] L. Boucher et al.: *TCP/IP offload network interface device @http://www.google.com/patents/US6434620*, August 2002.

[35] L. Deri: *ntop @http://www.ntop.org/*, Februar 2015.

[36] Linux Foundation: *GSO (Generic Segmentation Offload) @http://www.linuxfoundation.org/collaborate/workgroups/networking/gso*, März 2015.

[37] Linux Foundation: *UFO (UDP Fragmentation Offload) @http://www.linuxfoundation.org/collaborate/workgroups/networking/ufo*, März 2015.

[38] M. Blokzijl-Zanker: *Python sockets - sending string in chunks of 10 bytes @http://stackoverflow.com/a/13521333/1298153*, November 2012.

[39] M. Brown: *arping @http://linux-ip.net/html/tools-arping.html*, März 2015.

[40] M. Haardt et al.: *Linux Programmer's Manual, open, @http://man7.org/linux/man-pages/man2/open.2.html*, Mai 2015.

[41] M. Mitchell et al.: *Advanced Linux Programming @http://www.advancedlinuxprogramming.com/alp-folder/advanced-linux-programming.pdf*, Mai 2001.

[42] M. Rendold et al.: *Parallel Redundancy Protocol (PRP) @http://wiki.wireshark.org/PRP*, Januar 2013.

[43] M. Rentschler: *The Parallel Redundancy Protocol for Industrial IP Networks @http://ieeexplore.ieee.org/iel7/6495638/6505636/06505877.pdf*, Februar 2015.

[44] P. Biondi: *Scapy @http://www.secdev.org/projects/scapy*, April 2015.

[45] R. Faith et al.: *Linux Programmer's Manual, lseek, @http://man7.org/linux/man-pages/man2/lseek.2.html*, Juni 2014.

[46] R. Jones: *netperf @http://www.netperf.org/netperf/*, Februar 2015.

[47] S. Godard: *sysstat @http://sebastien.godard.pagesperso-orange.fr*, Februar 2015.

[48] S. Malssen et al.: *proc(5) - Linux man page @http://linux.die.net/man/5/proc*, März 2015.

[49] T. Herbert et al.: *Scaling in the Linux Networking Stack @https://www.kernel.org/doc/Documentation/networking/scaling.txt*, März 2015.

[50] T. Koenig et al.: *Linux Programmer's Manual, assert, @http://man7.org/linux/man-pages/man3/assert.3.html*, März 2015.

[51] T. Koenig et al.: *Linux Programmer's Manual, free, @http://man7.org/linux/man-pages/man3/free.3.html*, April 2015.

[52] T. Koenig et al.: *Linux Programmer's Manual, malloc, @http://man7.org/linux/man-pages/man3/malloc.3.html*, April 2015.

[53] TCPDUMP-Team: *TCPDUMP/LIBPCAP public repository @http://www.tcpdump.org*, Februar 2015.

[54] Technical Committee 65C - Industrial Networks: *Industrial communication networks - High availability automation networks - Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR) - Draft*, Februar 2015.

[55] V. Gropp: *bwm-ng (Bandwidth Monitor NG) @http://www.gropp.org/?id=projects&sub=bwm-ng*, April 2015.

[56] Wikipedia: *Memory Mapped I/O @http://de.wikipedia.org/wiki/Memory_Mapped_I/O*, Oktober 2013.

[57] Wikipedia: *Large segment offload @http://en.wikipedia.org/wiki/Large_segment_offload*, März 2015.

[58] Worcester Polytechnic Institute: *Fast Ethernet and Gigabit Ethernet @http://web.cs.wpi.edu/~rek/Undergrad_Nets/B05/Fast_Ethernet.ppt*, April 2015.

7 Glossar

Autonegotiation Ist ein Ethernet-Mechanismus, der es zwei miteinander verbundenen Geräten ermöglicht, automatisch die optimalen Grundeinstellungen bezüglich Geschwindigkeit, Duplex-Mode und Flow Control auszuhandeln. Während der Verhandlung teilen sich die beiden verbundenen Geräte ihre Eigenschaften mit und wählen dann die optimalen Einstellungen aus, welche beide Geräte unterstützen. Dieses Prozedere findet zu Beginn eines Verbindungsaufbaus zweier via Ethernetlink verbundenen Geräten statt.

Avg Kurzform für Average, zu Deutsch Durchschnitt

C-Makros Ein Makro ist ein benanntes Code-Fragment und wird meist in einer Header-Datei definert. Der Name des Fragments kann beliebig in Quellcode-Dateien platziert werden. Der C-Präprozessor ersetzt vor dem Kompilieren die Fragment-Namen durch die Code-Fragmente.

C-Pointer Ein Pointer in der Programmiersprache C, repräsentiert eine Speicheradresse.

Context Switch Prozedere, welches der Scheduler eines Betriebssystems durchführt, wenn ein neuer oder wartender Prozess CPU-Zeit erhält. Der aktuell auf der CPU laufende Prozess muss alle Kontextinformationen speichern und seine Ausführung unterbrechen, wobei der neue oder wartende Prozess seine Kontextinformationen laden und die Ausführung auf der CPU beginnen muss.

CRC **Cyclic Redundancy Check** ist ein Verfahren, basierend auf der modularen Polynomdivision (mod 2), welches dazu verwendet wird, einen Prüfwert über ein bestimmtes Datenfeld zu berechnen.

DAN **Double Attached Node oder Dual Attached Node**. Netzwerkteilnehmer eines Netzwerks, der über zwei Netzwerkinterfaces verfügt, die je direkt an einem LAN des Netzwerks angeschlossen sind.

DANP **Double Attached Node implementing PRP oder Dual Attached Node implementing PRP**. DAN in einem PRP-Netzwerk.

ECI **Ethernet Cable Interceptor**. Gerät für gezielte und reproduzierbare Störungen im Netzwerk, das bei einer Netzverbindung dazwischen geschaltet wird.

ethtool Software, die dazu verwendet werden kann, Gerätetreiber- und Hardware-Parameter von kabelgebundenen Netzwerk-Interfaces auszulesen, respektive zu modifizieren.

FCS Die **Frame Check Sequence** befindet sich am Ende eines Ethernet Frames. Sie hat eine Grösse von 4 Byte und wird von der sendenden Hardware (Ethernet-Controller) basierend auf dem Frame-Inhalt meist mittels CRC-Verfahrens berechnet und anschliessend am Ende eines jeden Frames platziert. Die empfangende Hardware berechnet für jedes empfangene Frame ebenfalls eine Frame Check Sequence und vergleicht diese mit der FCS am Ende des empfangenen Frames. Wenn beide FCS übereinstimmen, wird das Frame dem Netzwerk-Protokoll-Stack des Betriebssystems weitergereicht. Im Falle einer Divergenz der beiden FCS wird das Frame verworfen.

Gateway Netzwerkkomponente, die es ermöglicht, Computern in einem lokalen Netzwerk, den Zugriff in andere Netzwerke zu ermöglichen. Als bekanntestes Beispiel, kann an dieser Stelle die Verbindung von Computern in einem lokalen Netzwerk über einen Router als Gateway mit dem Internet erwähnt werden.

Kernelspace Bereich im Arbeitsspeicher, welcher ausschliesslich für Prozesse des Kernels reserviert ist. In diesem Bereich werden Kernelprozesse sowie privilegierte Systemprozesse abgelegt, respektive ausgeführt.

LRE **High Availability Seamless Redundancy**. Redundanzprotokoll für Ethernet basierte Netzwerke. HSR ist für redundant gekoppelte Ringtopologien ausgelegt. Die Datenübermittlung innerhalb eines HSR-Rings ist im Fehlerfall gewährleistet, wenn eine Netzwerkschnittstelle ausfallen sollte.

LRE **Link Redundancy Entitiy**. Einheit, die beide Netzwerkinterfaces eines DANs oder einer RedBox verbindet. Ist zuständig für die Frameduplikation und Duplikaterkennung.

Max Kurzform für Maximum

Memory Map Einteilung in verschiedene Segmente. Die kleinste adressierbare Einheit eines Segments ist ein Byte.

Memory Mapped I/O Ist ein Verfahren zur Kommunikation einer Zentraleinheit mit Peripheriegeräten. Die I/O-Register von elektronischen Bauelementen, mit denen angeschlossene Hardware gesteuert wird, werden in den Hauptspeicher-Adressraum abgebildet. Der Zugriff auf die Bauelemente kann dann über übliche Speicherzugriffsroutinen geschehen. Es werden keine besonderen Befehle benötigt wie bei der Realisierung der Ein-/Ausgabe mittels I/O-Ports am Prozessor. Gegenüber einem separaten I/O-Bus besitzt Memory Mapped I/O den Vorteil, dass man in der Regel über Strukturen und Pointer aus einer Hochsprache wie C oder C++ vollständig auf die Hardware zugreifen kann, ohne Teile des Programms in Assembler bzw. Maschinensprache schreiben zu müssen. [56]

Min Kurzform für Minimum

Netzwerk-Interface Bezeichnung für Netzwerkkarte / Netzwerk-Hardware.

NIC **Network Interface Card** / Netzwerkkarte.

Non-Free-Firmware Bei Non-Free-Firmware handelt es sich um Gerätetreiber-Software, die nicht als komplett freie Software vertrieben wird. Das heisst, die Software ist proprietär und basiert auf herstellerbasierten Standards, die nicht veröffentlicht wurden.

Payload Als Payload wird der reine Nutzdaten-Anteil eines Frames (Datalink-Layer) oder eines Paketes (Network-Layer, Transport-Layer, ...) definiert. Die Grösse des Payloads ist die Differenz der Gesamtpaketgrösse und der Grösse des entsprechenden Protokoll-Headers.

Programm-Header-Datei Eine Programm-Header-Datei hat die Dateiendung .h und enthält C-Funktionsdeklarationen sowie C-Makro-Definitionen. Die Header-Datei dient als Schnittstelle, zu den Quellcode-Dateien. Aufrufe von Funktionen, die in anderen Quellcodedateien implementiert wurden, werden durch das Einbinden der ensprechenden Header-Datei für den Compiler sichtbar.

Protokoll-Header Der Protokoll-Header enthält alle nötigen Metainformationen, die ein bestimmtes Kommunikationprotokoll braucht, um Daten austauschen zu können. Beispielfelder eines Protokoll-Headers: Quell- und Zieladresse, Port, Fragment-ID, etc.

PRP **Parallel Redundancy Protocol**. Hochverfügbarkeitsnetzwerk, bei dem Netzwerkkomponenten über zwei voneinander unabhängige LANs kommunizieren. Beim Versand wird das Frame dupliziert und über beide LANs versandt. Das Duplikat wird vom Empfänger erkannt und verworfen.

RCT **Redundancy Control Trailer**. 4 Bytes langes Framefeld, um Frames, die über beide LANs eines PRP-Netzwerks verschickt werden, zu kennzeichnen.

RedBox **Redundancy Box**. Ist mit beiden LANs des PRP-Netzwerks verbunden und bietet Anschlüsse für mehrere Hosts, damit diese über je 1 Netzwerkanschluss am PRP-Netzwerk teilnehmen können. Solche Hosts werden dann VDAN genannt.

RSS **Resident Set Size**. Beschreibt wie viel Speicher ein Prozess alloziert hat, der im Arbeitsspeicher ist. Zur RSS gehören zudem der Speicher, den Shared-Libraries des Prozesses im RAM belegen und den gesamten Stack- und Heap-Memory.

RTT **Round Trip Time** zu Deutsch Paketumlaufzeit. Damit ist die Zeit gemeint, welche benötigt wird, um von einer Quelle zum Ziel und wieder zurück zur Quelle zu gelangen.

RX **Reciever** / Empfänger.

SAN **Single Attached Node**. Host, der nur an einem LAN des PRP-Netzwerks angeschlossen ist. Dieser kann mit allen DANs, VDANs und RedBoxen kommunizieren, jedoch nur mit anderen SANs, die am selben LAN angeschlossen sind. Ist zum Beispiel ein SAN nur am LAN A angeschlossen, kann dieser nur andere SANs erreichen, die auch am LAN A angeschlossen sind.

Socket Sockets bilden das Bindeglied zwischen Kommunikationsprotokollen und dem Betriebssystem. Ein Socket kann auch als Endpunkt einer Verbindung angesehen werden. Da in einem unixoiden System das Konzept gilt, ”Alles ist eine Datei.”, wird ein Socket seitens Betriebssystem als Datei repräsentiert. Der sendende Prozess schreibt die Daten in seine entsprechende Socket-Datei, wobei diametral dazu ein empfangender Prozess die Daten von seiner entsprechenden Socket-Datei liest.

System-Call Anfrage einer Applikation an den Kernel des Betriebssystems.

TX **Transmitter** / Sender.

Userspace Bereich im Arbeitsspeicher, wo normale Benutzerprozesse abgelegt respektive ausgeführt werden.

VDAN **Virtual Double Attached Node oder Virtual Dual Attached Node**. Host, der über ein Netzwerkinterface an einer RedBox angeschlossen ist und somit darüber am PRP-Netzwerk teilnimmt. Für andere Netzwerkteilnehmer wird dieser Host wie ein DAN wahrgenommen.

VSZ **Virtual Memory Size**. Beschreibt den gesamten Speicher, der von einem Prozess benutzt wird, inklusive dem Speicher ausserhalb vom RAM und Shared-Libraries, die der Prozess verwendet.

Zero-copy Modus Zero-copy beschreibt Computer-Operationen, bei denen die CPU nicht dafür zuständig ist, die Daten vom einen Speicher auf den anderen zu kopieren. Solche Operationen werden gebraucht, um beim Senden von Daten über ein Netzwerk an Prozessorleistung und Arbeitsspeicher-Gebrauch zu sparen.

8 Abbildungsverzeichnis

9 Tabellenverzeichnis

10 Listingverzeichnis

Teil IV

Anhang

11 Offizielle Aufgabenstellung

Nachfolgend ist die offizielle Aufgabenstellung [22] platziert, welche den Autoren am 09.02.2015 zugestellt wurde:

Hans Weibel
Büro: TW 219
Telefon:058 / 934 75 52
E-Mail: hans.weibel@zhaw.ch
WWW: http://www.zhaw.ch/~wlan/

BA15_wlan_1: Projektarbeit
im Fachgebiet Kommunikation

Ermittlung der Performance von Netzwerkfunktionen am Beispiel von PRP

Bearbeitung durch: Mauro Guadagnini und Prosper Sebastian Leibundgut
Betreuung durch: Prof. Hans Weibel
Partner: Institute of Embedded Systems der ZHAW
Ausgabe: Montag, 9. Februar 2015 / Abgabe: 5. Juni 2015

1 Ausgangslage

Will man die Leistungsfähigkeit der Netzwerkanbindung eines Rechners beurteilen, bzw. den Ressourcenbedarf zur Erzielung einer bestimmten Leistung ermitteln, so bewegt man sich in einem sehr komplexen Feld. Die beobachtbare Leistung ergibt sich aus dem Zusammenspiel von Rechenleistung, Betriebssystem, Protokollsoftware, Netzwerkadapter, Netzwerk, Anwendungssoftware, Verhalten des Kommunikationspartners und zu guter Letzt dem Protokoll selbst. Es ist nicht immer einfach zu erkennen, welcher Faktor limitierend wirkt. Darüber hinaus ist zu berücksichtigen, dass Beobachtungs- und Messverfahren das Verhalten mitbeeinflussen können.

In der vorliegenden Arbeit sollen derartige Fragestellungen vorerst an einem konkreten Fall studiert und anschliessend verallgemeinert werden. Beim konkreten Fall handelt es sich um ein Ethernet-Redundanzprotokoll, das auf dem Host alleine implementiert wird (d.h. dass keine Unterstützung vom Netzwerk selbst geleistet werden muss). Es handelt sich um das vom InES mitentwickelten PRP (Parallel Redundancy Protocol). PRP ermöglicht eine unterbrechungsfreie hochverfügbare Kommunikation, wie sie von gewissen sicherheitskritischen Anwendungen verlangt wird. Das Verfahren besteht darin, dass zwei unabhängige Netzwerke verwendet werden und die Endknoten doppelt angebunden werden. Jedes Frame wird vom sendenden Knoten repliziert und auf beiden Netzwerken übertragen. Der Empfänger verarbeitet das zuerst eintreffende Frame und verwirft das Duplikat. Dazu wird in den End-

Abbildung 11.1: Offizielle Aufgabenstellung, Seite 1

knoten ein Protokoll-Layer eingeführt, der vom InES als open Source verbreitet wird (siehe https://github.com/ZHAW-InES-Team/sw_stack_prp1). Es soll nun untersucht werden, wie sich die Anwendung des Verfahrens auf die Performance des Endknotens auswirkt bzw. welche zusätzlichen Ressourcen benötigt werden.

In der Verallgemeinerung soll aufgezeigt werden,

- wie Testverfahren und Hilfsmittel arbeiten und anzuwenden sind
- wie Testszenarien gestaltet sein müssen, dass die zu ermittelnden Eigenschaften der "Unit under Test" von den Einflüssen der Testumgebung unterschieden werden können
- wie die Resultate zu interpretieren sind

2 Aufgabenstellung

Im Zentrum der Arbeit stehen Linux-Endsysteme (kann jedoch in Bezug auf PRP auch auf Windows ausgedehnt werden).

Die untenstehende Auflistung von Teilaufgaben gibt keine Chronologie vor. Es werden vermutlich mehrere Zyklen durchlaufen, in welchen die einzelnen Aspekte verfeinert und konkretisiert werden.

2.1 Studium von PRP und Aufbau Testumgebung

Es ist eine einfache PRP-Umgebung aufzubauen, die aus zwei Netzwerken A und B (je durch einen Ethernet-Switch repräsentiert) sowie zwei bis drei Endsystemen bestehen. Das Protokoll und seine Implementierung sind zu studieren.

Es sollen Konfigurationen definiert werden, in welchen die Leistungsfähigkeit der Implementierung bzw. der zusätzlichen Bedarf von Ressourcen ermitteln werden können. Dabei sollen unter anderem Aspekte beachtet werden wie:

- Effekt von Laufzeitunterschieden zwischen Netz A und Netz B (konstante und schwankende Unterschiede, sprunghafte Änderungen).
- Zeitweiser Ausfall eines Netzwerkpfades.
- Abhängigkeit vom verwendeten Protokoll (TCP, UDP).
- Einfluss nicht entfernter Duplikate auf Funktion und Performance.
- Auswirkung auf Applikationen, wenn Frames out-of-sequence ankommen (wenn z.B. auf dem kürzeren Pfad Frames verloren gehen oder wenn Frames priorisiert werden).

2.2 Untersuchung von Tools

Es können zwei Ansätze unterschieden werden: Blackbox- und Whitebox-Messungen.

Für den Blackbox-Ansatz existieren diverse Tools, welche die Nutzung von Ressourcen bzw. Performanceparameter messen (z.B. ping, BWping, iperf, netperf, flowgrind, top, ntop, atop, sar, und viele andere mehr).

Voraussetzung für den Whitebox-Ansatz ist der Zugriff auf den Programmcode der zu untersuchenden Funktion. Mittels Profiling-Tools kann man einen Einblick in das Ausführungsverhalten des Codes gewinnen.

Neben Linux-Bordmitteln sind auch selbst entwickelte Hilfsmittel (z.B. Applikationen, die eine genau definierte Last erzeugen) und dedizierte Messgeräte in Betracht zu ziehen.

Abbildung 11.2: Offizielle Aufgabenstellung, Seite 2

Mit Hilfe von geeigneten Kalibrier- und Vergleichsmessungen soll aufgezeigt werden, was die ermittelten Resultate aussagen bzw. wie sie zu interpretieren sind.

Es soll auch geklärt werden, welchen Einfluss die Offload-Mechanismen moderner Ethernet-Adapter haben.

2.3 Verhalten der PRP Implementierung

Es ist zu studieren, wie sich die Anwendung von PRP auf das Endsystem auswirkt, insbesondere auf die Belastung von CPU und Memory. Dabei sollen dual und single attached Endknoten verglichen werden.

2.4 Generalisierung

Die Messungen und die Interpretation der Resultate werden nur dann eine Aussagekraft haben, wenn gewisse Grundsätze und Rahmenbedingungen eingehalten sind. Diese Grundsätze und Rahmenbedingungen sind zu formulieren. Ebenso soll eine Bewertung der untersuchten Tools abgegeben werden.

2.5 Weiterführende Aspekte

Die zunehmende Virtualisierung von Processing, Storage und Network bringt eine neue Dimension in diese Betrachtungen. Wie sieht es aus, wenn Virtuelle Maschinen auf demselben Host miteinander kommunizieren? Was leisten die betrachteten Tools in dieser Umgebung?

3 Ziele

- Für einige ausgewählte Tools und Messmethoden herrscht Klarheit, welche Messungen sie ermöglichen und wie die Resultate zu interpretieren sind.
- Es liegt ein PRP- Testnetzwerk vor, welches erlaubt, die relevanten Szenarien auszumessen.
- Die InES-Implementierung von PRP ist in Bezug auf Leistungsfähigkeit und Ressourcenbedarf evaluiert.
- Es sollen einige grundsätzliche Aussagen gemacht werden darüber, wie und mit welcher Zuverlässigkeit man die Performance in physikalischen und virtuellen Konfigurationen messen kann.

Abbildung 11.3: Offizielle Aufgabenstellung, Seite 3

12 Projektmanagement

12.1 Präzisierung der Aufgabenstellung

12.2 Besprechungsprotokolle

Die Besprechungsprotokolle wurden Stichwortartig in einem eigenen Wiki festgehalten. Der Inhalt dieser Protokolle lautet wie folgt:

12.2.1 Kalenderwoche xx: xx.xx.2015

- Lorem ipsum

13 Einrichtung ECI

In diesem Kapitel wird beschrieben wie der ECI in dieser Arbeit eingerichtet wird. Weitere Einrichtungsmöglichkeiten und Informationen zum ECI können in dessen Bedienungsanleitung [6] eingesehen werden.

Der ECI wird in dieser Arbeit auf einem Laptop mit einem Linux-Betriebssystem (Distribution: ArchLinux) operiert. Die Angaben zur Installation können für andere Computer variieren. Der Treiber, um den ECI ansprechen zu können, ist im Linux Kernel ab Version 3.0.0-19 implementiert. [16]

13.1 Installation der Software

Um die Software bedienen zu können, wird eine Java-Library für serielle Kommunikation, die von `http://rxtx.qbang.org/wiki/index.php/Download` (Datei: `rxtx-2.1-7-bins-r2.zip`) heruntergeladen werden kann, benötigt. Ist die Datei `rxtx-2.2pre2-bins-r2.zip` entpackt, kopiert man `RXTXcomm.jar` nach `/usr/lib/jvm/java-7-openjdk/jre/lib/ext` und `x86_64-unknown-linux-gnu/librxtxSerial.so` nach `/usr/lib/jvm/java-7-openjdk/jre/lib`

Des Weiteren muss der Benutzer, mit dem die Software ausgeführt wird, in der Benutzergruppe `lock` sein. Diesen für man mit folgendem Befehl dieser Gruppe hinzu: `sudo usermod -a -G lock $USERNAME`

14 Quellcode

14.1 meas - Messung von CPU- und Netzwerk-Last

14.2 shck - Netzwerklast-Generierung

14.3 Szenarien

14.3.1 Szenario 01: Performance im PRP-Netzwerk