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	Hans Weibel (wlan@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:

……...

……

……

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Bachelorarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

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. [41]

PRP ist bereits bei einigen Firmen implementiert und wird unter anderem für Substation Automation verwendet. [3, 18]

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 [3] oder Verbesserungen vorschlagen, welche in einem eventuell zukünftigen PRP-2 implementiert werden könnten [34].

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 [16] (siehe Kapitel 11 auf Seite 61) zugestellt, welche die nachfolgenden Hauptanforderungen umfasst:

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. [41]

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 15 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. [15]

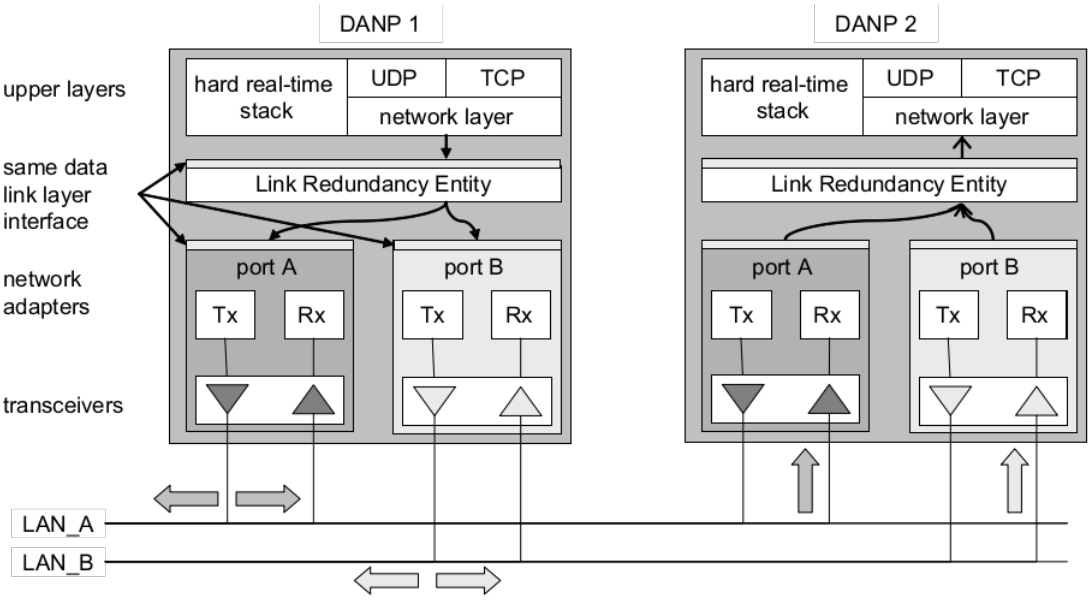

Abbildung 2.1.: PRP-Netzwerk mit 2 kommunizierenden DANPs [41]

Dieser RCT hat eine Grösse von 6 Bytes und beeinhaltet folgende Parameter [41]:

- Sequenznummer (16 Bit)
  Jede Quelle hat lediglich einen Sequenznummernraum [21]
- 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 [34] 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.

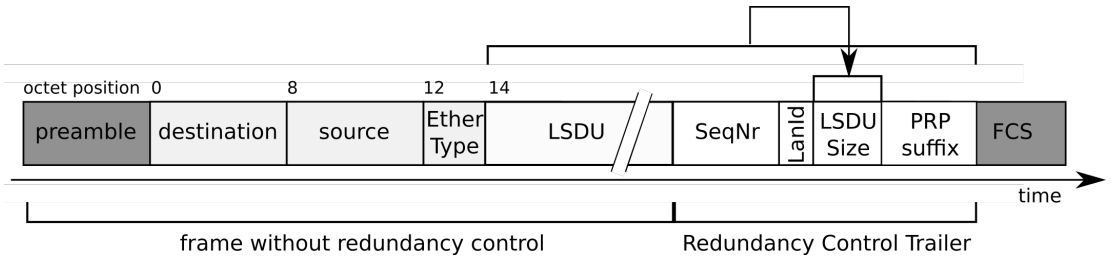

Abbildung 2.2.: PRP-Frame mit RCT [41]

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. [18]

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. [15, 18]

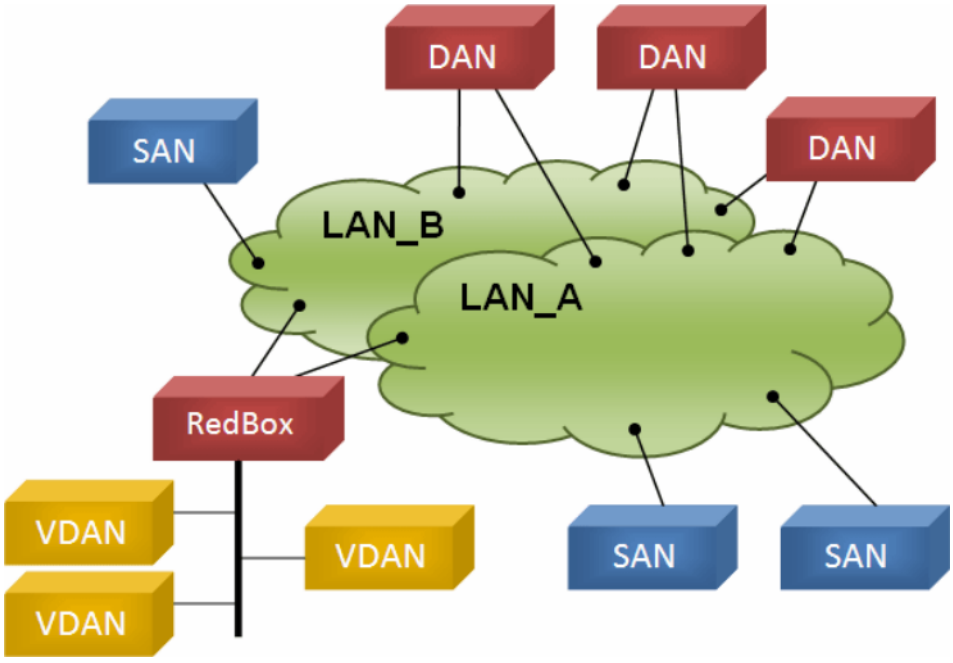

Abbildung 2.3.: Beispielaufbau eines PRP-Netzwerks [15]

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. [17]

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. [34]

Für das Redundanzmanagement und das Überprüfen, ob andere DANs im Netzwerk anwesend sind, sendet jede LRE via Multicast regelmässig (laut Standard [41] alle 400 ms) so-

genannte «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. [41]

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). [41]

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. [20]

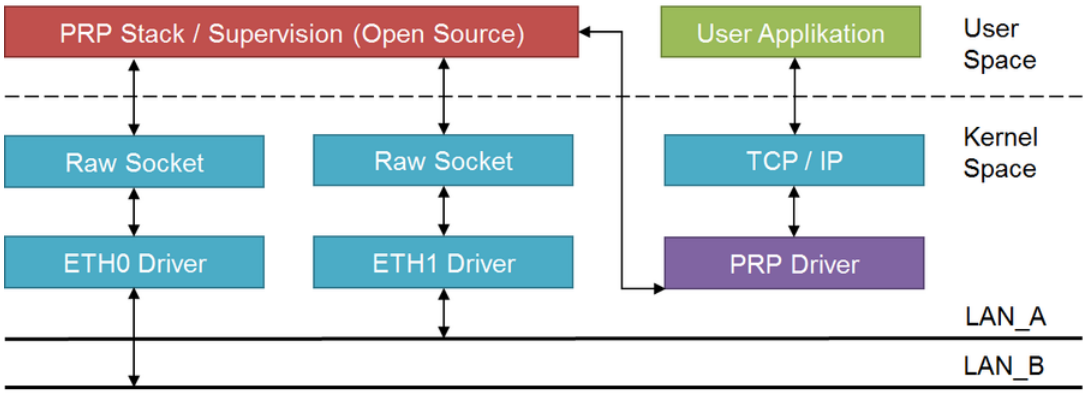

Abbildung 2.4.: PRP-1 User Mode Stack [20]

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. [12]

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. [9]

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. [42]

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. [30]

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. [22, 29, 42]

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. [22]

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. [22]

RX/TX VLAN Acceleration

Mittels VLAN Acceleration wird beim Senden / Empfangen der VLAN-Header in der NIC hinzugefügt / gekürzt. [24]

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. [7, 39]

RX Hashing Offload

Anhand von «RX Hashing Offload» wird bei Erhalt eines Frames bereits in der Netzwerkkarte die Prüfsumme automatisch verifiziert. [27]

2.3. /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. [38]

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. [32]

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. Nichtprozessbezogene Informationen findet man im Wurzelverzeichnis des /proc-Dateisystems. Zum Beispiel befinden sich detaillierte Informationen zum Arbeitsspeicher unter /proc/meminfo. [32]

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 16) sind deaktiviert, wobei dessen Einflüsse später untersucht werden (siehe Kapitel 3.3.4.9 auf Seite 46).

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
Netzwerkanschluss «eth1»	1 GBit/s
Netzwerkanschluss «eth2»	100 MBit/s
Betriebssystem	Debian 7.8.0 32bit (i686 / x86 CPU) Version mit Non-Free-Firmware inklusive

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

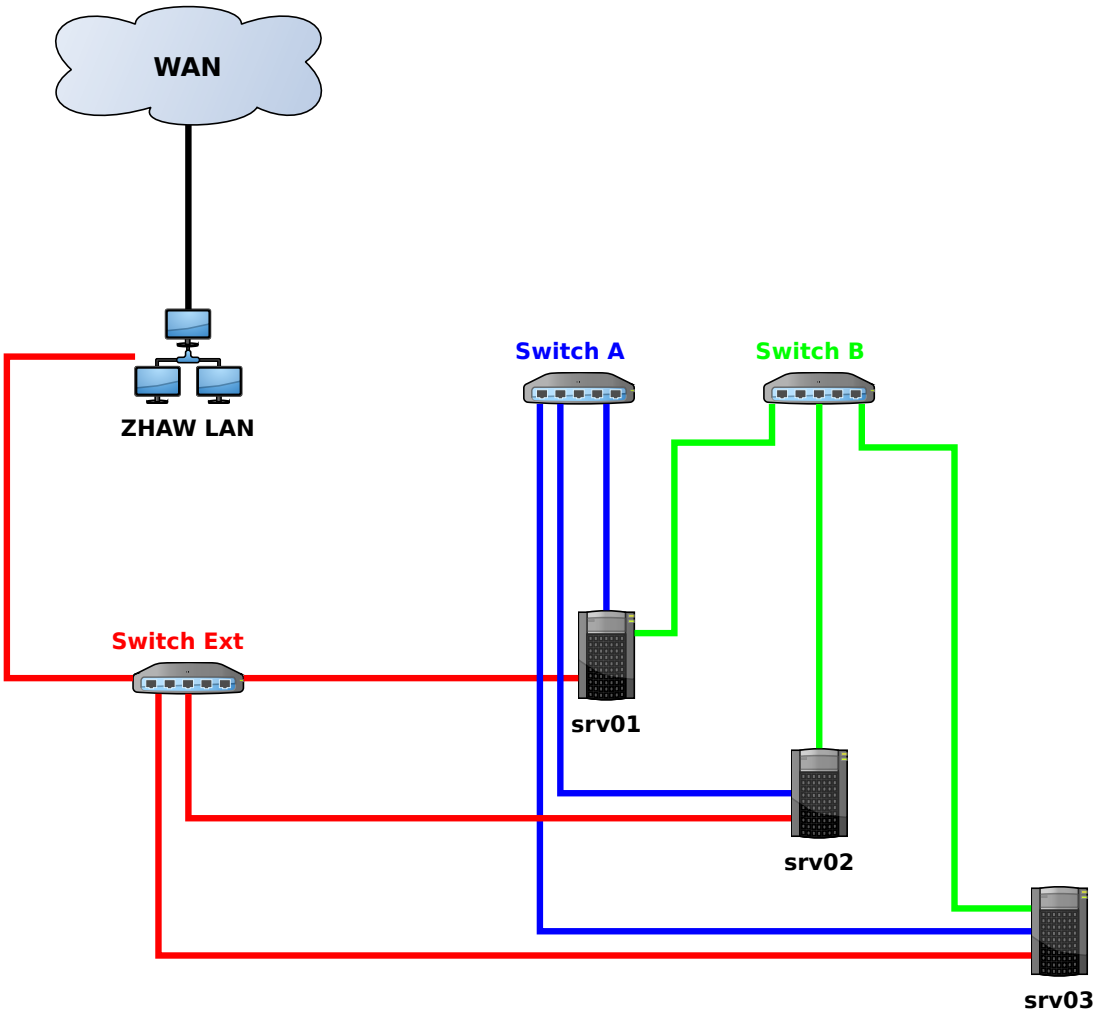

Abbildung 3.1.: Aufbau der Testumgebung - Physisches Netzwerk

	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.

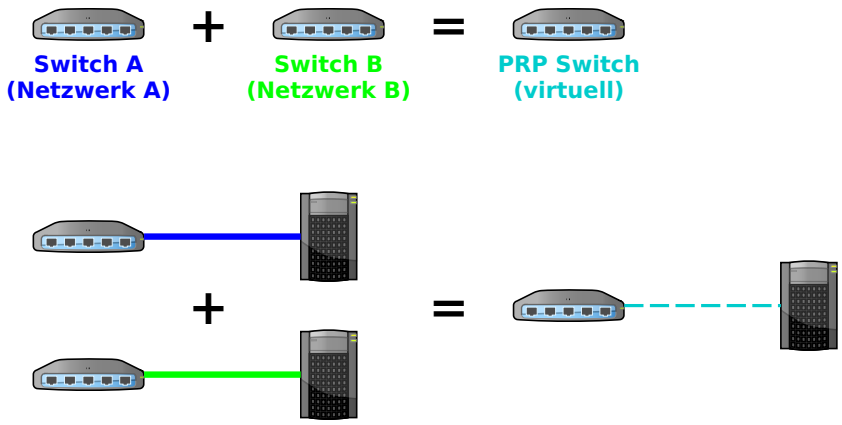

Abbildung 3.2.: Zusammenhang physische und virtuelle Verbindung

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

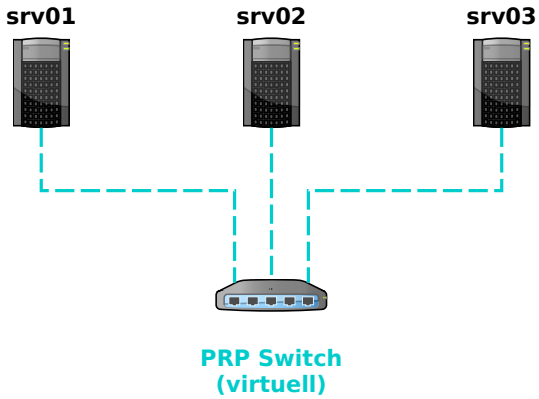

Abbildung 3.3.: Aufbau der Testumgebung - PRP-Netzwerk

	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

IPv4-Konfiguration:

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:

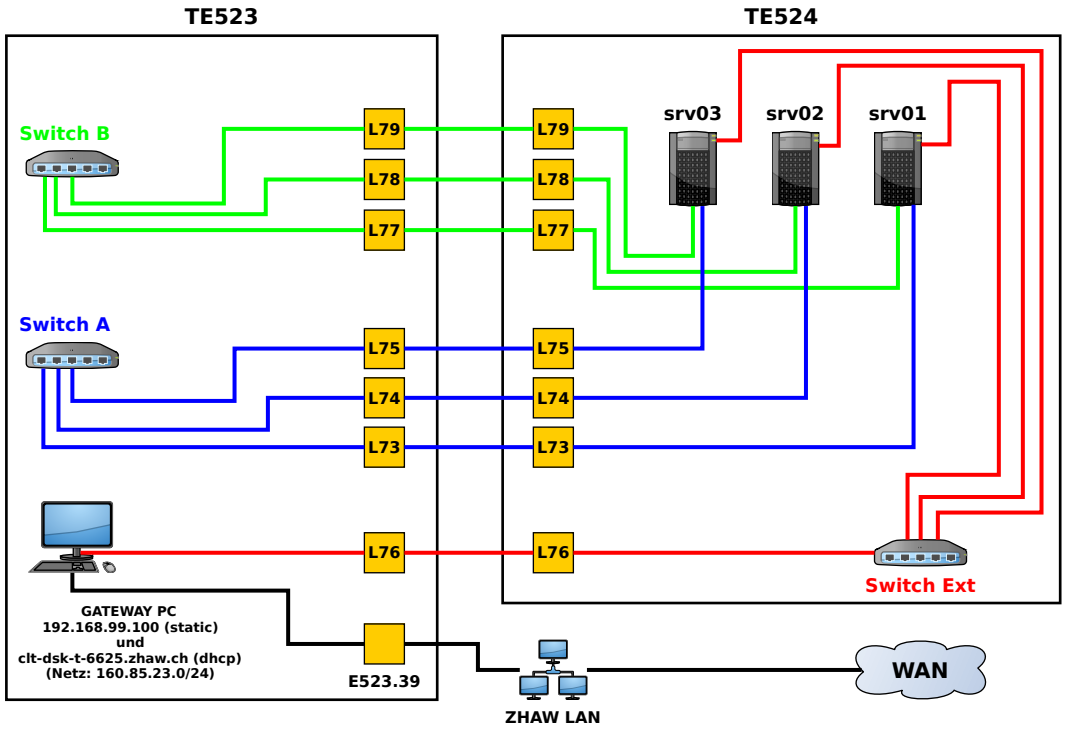

Abbildung 3.4.: Aufbau der Testumgebung - Detaillierte Verkabelung

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 [16] 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.

3.2.1. Generierung von Netzwerktraffic

Name	**flowgrind [6]**
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 [16] 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 [8]**
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 [36]**
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 [13]**
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	**NetHogs [4]**
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 [4] 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 3.8.: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: NetHogs

Name	**ntop [28]**
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 3.9.: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: ntop

Name	**pmap [1]**
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 3.10.: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: pmap

Name	**strace [23]**
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 15) und dem Betriebssystem untersucht werden.

Tabelle 3.11.: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: strace

Name	**sysstat [37]**
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.

Tabelle 3.12.: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: sysstat

Name	**tcpdump [40]**
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 3.13.: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: tcpdump

Name	**top [2]**
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 3.14.: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: top

Name	**tshark [11]**
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. [33]

Tabelle 3.15.: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: tshark

Name	**valgrind [26]**
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 3.16.: Hilfsmittel zum Messen von Ressourcennutzung und Performanceparametern: valgrind

3.2.3. Weitere Tools

Name	**arping [31]**
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 3.17.: Weitere Hilfsmittel: arping

Name	**Ethernet Cable Interceptor (ECI) [5, 19]**
Beschrieb	Kleine Box, welche bei einem 10/100 Base-TX Netzwerkanschluss dazwischen geschaltet werden kann und benötigt somit weder Modifikationen an Hardware oder zusätzlichen Treibern.
Funktionen	Mit dem ECI wird ein Delay und falls gewünscht ein Frameverlust erzeugt. 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 3.18.: Weitere Hilfsmittel: Ethernet Cable Interceptor (ECI)

Name	**vmstat [14]**
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 3.19.: 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») programmiert, bei der die CPU-Last berechnet wird, um die Ergebnisse von pidstat und top verifizieren zu können. Des Weiteren misst meas neben der CPU-Last auch die verwendete Netzwerkbandbreite. Die Applikation wie folgt:

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 [35] 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 TCP-Pakete, welche permanent mit der minimalen Grösse versendet werden oder das Versenden von reinen Ethernet-Frames mit zufälligen und dennoch reproduzierbaren Grössen).

Der Quellcode ist der Arbeit beigelegt und wird dort auch genauer erörtert (siehe Kapitel 14.2 auf Seite 66). In diesem Abschnitt gibt es eine grobe 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.
-i	Interface Bestimmt, über welches Netzwerkinterface die Netzwerklast versendet werden soll.
-m	MTU Bestimmt die MTU (Standard-Wert beträgt 1500)
-P	PRP-Modus (benötigt kein Argument) Wird dieser Parameter mit angegeben, so wird der Nutzlast 6 Bytes 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.
-s	Sizetype / Grössentyp Bestimmt den Grössentyp der Netzwerklast. Zur Auswahl stehen «MIN», «MAX» und «RANDOM». Bei «MIN» und «MAX» besteht die Netzwerklast jeweils aus lauter kleinen oder grossen Frames / Paketen. Mit «RANDOM» als Argument erhalten die Frames / Pakete jeweils eine zufällige Grösse. Diese Grössen werden aus einer Datei ausgelesen, die mit Zufallswerten zwischen 64 und 1500 generiert wurde. So kann eine zufällige Netzwerklast generiert werden, die dennoch reproduzierbar ist.
-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 3.20.: Parameter der Applikation «shck»

3.2.5. Verwendete Tools

In diesem Kapitel werden die in dieser Arbeit verwendeten Tools aufgelistet. Dazu wird aufgeführt welche Aufgabe von welcher Applikation übernommen wird.

Tool	Aufgabe in dieser Arbeit
arping	Überprüfen der Erreichbarkeit der Computer im Netzwerk auf Layer 2 mittels ARP
ECI	Erzeugen von Störungen im Netzwerk
iperf3	Generieren und Versenden von Netzwerktraffic über TCP und UDP
meas	Performance-Ermittlung des PRP-1 Stacks
pidstat (sysstat)	Analyse des PRP-1 Stacks (siehe Kapitel 2.1.1 auf Seite 15) im Betrieb
pmap	Festellen des Speichers, der vom PRP-1 Stack beansprucht wird
shck	Netzwerklast-Generierung für alle Testfälle
strace	Anzeigen von System-Calls des PRP-1 Stacks, um dessen Verhalten nach zu vollziehen
top	Performance des PRP-1 Stacks bezüglich CPU und RAM überwachen
tshark	Festhalten und Analysieren des Netzwerkverkehrs
valgrind	Fehlersuche in eigens entwickelten C-Applikationen

Tabelle 3.21.: Verwendete Tools und ihre Aufgaben in dieser Arbeit

3.2.6. Zuverlässigkeit der Messwerte

Die Zuverlässigkeit der Messwerte wird überprüft, indem Messungen mit unterschiedlichen Tools nacheinander bei den gleichen Bedingungen ausgeführt. So wird vermieden, dass sich die Tools für die Messungen gegenseitig beeinflussen. Die Bedingungen werden in den folgenden Kapiteln beschrieben.

3.2.6.1. CPU

Auf einem der DANs wird iperf3 (siehe Tabelle 3.5 auf Seite 26) 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. 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.

```
#!/bin/sh
(dd if=/dev/urandom of=/tmp/randomfile bs=1M count=10000 &)
pid_dd=$(ps -ef | grep -w dd | cut -d' ' -f3 | head -n1)
(./pidstat.sh dd &)
(./top.sh dd $pid_dd &)
sleep 15
pkill dd
(iperf3 -c 192.168.0.2 -d -n 200M &)
(./pidstat.sh iperf3 &)
(./top.sh iperf3 &)
```

Listing 3.1: Bash-Script zur Zuverlässigkeitsüberprüfung der CPU-Messungen

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

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

Listing 3.2: Bash-Script zur Zuverlässigkeitsüberprüfung der CPU-Messungen von pidstat (cpu/pidstat.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
                sleep 5
                top -b -n 2 -d 5.0 -p $2 > results/top_dd.txt
                ;;
        "iperf3")
                sleep 5
        top -b -n 2 -d 5.0 -p $(pgrep iperf3) > results/top_iperf3.txt
                ;;
        *)
                echo "only use dd or iperf3 as parameter"
                ;;
esac
```

Listing 3.3: Bash-Script zur Zuverlässigkeitsüberprüfung der CPU-Messungen von top (cpu/top.sh)

Folgende Werte werden von pidstat, top und meas ausgegeben:

Tool	%CPU (5s nach Start)	%CPU (10s nach Start)
meas-Messung von dd		
pidstat-Messung von dd		
top-Messung von dd		
meas-Messung von iperf3		
pidstat-Messung von iperf3		
top-Messung von iperf3		

Tabelle 3.22.: Zuverlässigkeitsüberprüfung der CPU-Messungen von meas, pidstat und top

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 CPU-Last 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 in dieser Arbeit verwendet.

pidstat

top

meas

3.2.6.2. Arbeitsspeicher

Wie im vorherigen Kapitel (siehe Kapitel 3.2.6.1 auf Seite 34) 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' ' -f3 | head -n1)
(./pidstat.sh dd &)
(./pmap.sh dd &)
(./top.sh dd $pid_dd &)
sleep 15
pkill dd

(iperf3 -c 192.168.0.2 -d -n 200M &)
(./pidstat.sh iperf3 &)
(./pmap.sh iperf3 &)
(./top.sh iperf3 &)
```

Listing 3.4: Bash-Script zur Zuverlässigkeitsüberprüfung der Arbeitsspeicher-Messungen

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

Listing 3.5: Bash-Script zur Zuverlässigkeitsüberprüfung 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/pmap_dd.txt
      sleep 5
      pmap -x $(pgrep dd) >> results/pmap_dd.txt
      ;;
    "iperf3")
      sleep 5
      pmap -x $(pgrep iperf3) > results/pmap_iperf3.txt
      sleep 5
      pmap -x $(pgrep iperf3) >> results/pmap_iperf3.txt
      ;;
    *)
      echo "only use dd or iperf3 as parameter"
           ;;
esac
```

Listing 3.6: Bash-Script zur Zuverlässigkeitsüberprüfung der Arbeitsspeicher-Messungen von pmap (ram/pmap.sh)

```
#!/bin/sh
if [ -z $1 ]
then
        echo "parameter dd or iperf3 needed"
        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                  sleep 5
16                  top -b -n 2 -d 5.0 -p $(pgrep dd) > results/top_dd.txt
17                  ;;
18          "iperf3")
19                  sleep 5
20          top -b -n 2 -d 5.0 -p $(pgrep iperf3) > results/top_iperf3.txt
21                  ;;
22          *)
23                  echo "only use dd or iperf3 as parameter"
24                  ;;
25  esac
```

Listing 3.7: Bash-Script zur Zuverlässigkeitsüberprüfung 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. [25]

Folgende Werte werden von pidstat, pmap und top ausgegeben:

Tool	**VSZ in Kilobytes (5s nach Start)**	**VSZ in Kilobytes (10s nach Start)**	**RSS in Kilobytes (5s nach Start)**	**RSS in Kilobytes (10s nach Start)**
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 3.23.: Zuverlässigkeitsüberprüfung 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.

3.2.6.3. Netzwerkbelastung

Im Kapitel zur Zuverlässigkeitsüberprüfungen der CPU-Messungen (siehe Kapitel 3.2.6.1 auf Seite 34) wird eine eigens entwickelte Applikation 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.

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 34 ausgeführt, jedoch erfolgt hier die Messung zum gleichen Zeitpunkt, an dem der iperf3-Client gestartet wird.

Das Bash-Skript sieht wie folgt aus:

```
1  #!/bin/sh
2  (iperf3 -c 192.168.0.2 -d &)
3  (meas &)
```

Listing 3.8: Bash-Script zur Zuverlässigkeitsüberprüfung der CPU-Messungen

	srv01 (iperf3-Client, tätigt Upload)				srv02 (iperf3-Server, tätigt Download)			
Interval [s]	**iperf3 TX Daten [MBytes]**	**iperf3 TX-Rate [MBits/s]**	**meas TX Daten [Bytes]**	**meas TX-Rate [Bits/s]**	**iperf3 RX Daten [MBytes]**	**iperf3 RX-Rate [MBits/s]**	**meas RX Daten [Bytes]**	**meas RX-Rate [Bits/s]**
0-1								
1-2								
2-3								
3-4								
4-5								
5-6								
6-7								
7-8								
8-9								
9-10								

Tabelle 3.24.: Zuverlässigkeitsüberprüfung der Netzwerkbelastungs-Messungen anhand von iperf3 und meas

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 47 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 20) 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.

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

Des Weiteren werden für die Generierung der Netzwerklast 3 verschiedene Typen angewandt:

Der Netzwerkverkehr besteht...

- **...nur aus kleinen Frames** (64 Bytes + 6 Bytes RCT = 70 Bytes, wenn ein VLAN-Tag gebraucht wird 74 Bytes [41])
  Hierbei handelt es sich um einen der schlimmsten Fälle, da weitaus weniger Zeit für die Duplikaterkennung vorhanden ist, bis das nächste Frame ankommt.
- **...nur aus grossen Frames** (Mit RCT 1528 Bytes, im PRP-1-Standard wird angenommen, dass die jede Netzwerkkomponente diese von ISO/IEC/IEEE 8802-3:2014 vorhergesehene «Oversize»-Grösse unterstützt [41])
  Der zweite der schlimmsten Fälle, da permanent grosse Daten verarbeitet werden müssen.

- **...aus Frames mit zufälliger Grösse**
  Dieser Netzwerkfluss stellt die Mitte zwischen den oberen beiden Typen dar. Es wird im Voraus eine grosse Liste mit zufälligen Zahlen generiert, die sich zwischen der kleinst- und grösstmöglichen Framegrösse befinden. Diese Liste wird bei der Generierung der Netzwerklast stets von Oben nach Unten abgearbeitet, um so zufällige Grössen, aber auch Reproduzierbarkeit garantieren zu können.

3.3.2. Wie wird gemessen?

Die Messungen werden in Form von Bash-Scripts und C-Applikationen vorbereitet (Quellcode siehe Kapitel 14 auf Seite 66), um automatisiert ausgeführt werden zu können.

Der Vorgang einer Messung lautet im Groben wie folgt:

1. Starten der Netzwerkgenerierung auf einem Rechner, sodass sie etwas länger als die gewählte Zeitspanne (Ultrakurz-, Kurz- oder Langzeit) Netzwerkverkehr stattfindet.

2. Kurz darauf (ca. 5 Sekunden) werden gleichzeitig die Messungen auf allen beteiligten Servern gestartet. Zum Startzeitpunkt der Messungen befindet man so sich sicher bereits an einem Punkt, an dem sich ein konstanter Zustand (Steady State) eingependelt hat.

3. Während der gewählten Zeitspanne wird die CPU-Last, der Arbeitsspeicherverbrauch 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.

4. Ist die Zeitspanne abgelaufen, wird die Messung 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.

Solch eine Messung wird in jedem Szenario jeweils für jede Zeitspanne (Ultrakurz-, Kurz- und Langzeit) ein mal durchgeführt. In den ermittelten Daten befinden sich Werte zur CPU-Last, Arbeitsspeicherbelegung und Netzwerkbelastung.

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 des Prozesses ergibt.

- **Arbeitsspeicher-Nutzung**
  Eine Berechnung der Werte ist hier nicht nötig, die RSS- und VSZ-Werte können direkt ausgelesen werden. So kann ermittelt werden wie viel Speicher der PRP-1 stack beansprucht und wie viel davon im Arbeitsspeicher und wie viel ausgelagert ist.

- **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.	%
Usertime		s
Systemtime		s
CPU-Last Maximum	Wert der Periode innerhalb der Zeitspanne, in der das untersuchte Objekt am meisten Zeit vom Prozessor beansprucht hat.	%
Usertime		s
Systemtime		s
CPU-Last Durchschnitt	Durchschnitt der Werte jeder Periode innerhalb der Zeitspanne. Besagt wie viel der Zeit innerhalb der Zeitspanne die CPU für das untersuchte Objekt aufgewendet hat.	%
Usertime		s
Systemtime		s
Arbeitsspeicher RSS Minimum	Kleinster RSS-Wert unter allen Perioden, der vom untersuchten Objekt im Arbeitsspeicher beansprucht wurde.	KBytes
Arbeitsspeicher RSS Maximum	Grösster RSS-Wert unter allen Perioden, der vom untersuchten Objekt im Arbeitsspeicher beansprucht wurde.	KBytes
Arbeitsspeicher RSS Durchschnitt	Durchschnittlicher RSS-Wert über die gesamte Zeitspanne, der vom untersuchten Objekt im Arbeitsspeicher beansprucht wurde.	KBytes
Arbeitsspeicher VSZ Minimum	Kleinster VSZ-Wert unter allen Perioden, der vom untersuchten Objekt im Arbeitsspeicher beansprucht wurde.	KBytes
Arbeitsspeicher VSZ Maximum	Grösster RSS-Wert unter allen Perioden, der vom untersuchten Objekt im Arbeitsspeicher beansprucht wurde.	KBytes
Arbeitsspeicher VSZ Durchschnitt	Durchschnittlicher RSS-Wert über die gesamte Zeitspanne, der vom untersuchten Objekt im Arbeitsspeicher beansprucht wurde.	KBytes

Tabelle 3.26.: Beschreibung der Werte vom Ergebnis einer Messung: CPU und Arbeitsspeicher

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
Netzwerkbelastung Download Minimum	Kleinste Netzwerkbelastung der Periode innerhalb der Zeitspanne, in der das Netzwerkinterface am wenigsten das Netzwerk belastet hat.	Bytes/s
Netzwerkbelastung Upload Minimum		
Netzwerkbelastung Download Maximum	Grösste Netzwerkbelastung der Periode innerhalb der Zeitspanne, in der das Netzwerkinterface am meisten das Netzwerk belastet hat.	Bytes/s
Netzwerkbelastung Upload Maximum		
Netzwerkbelastung Download Durchschnitt	Durchschnittliche Netzwerkbelastung über die ganze Zeitspanne, in der gemessen wurde.	Bytes/s
Netzwerkbelastung Upload Durchschnitt		
Erhaltene Bytes Minimum	Kleinste Anzahl an erhaltenen/übermittelten Bytes der Periode innerhalb der Zeitspanne, in der das Netzwerkinterface am wenigsten das Netzwerk belastet hat.	Bytes
Übermittelte Bytes Minimum		
Erhaltene Bytes Maximum	Grösste Anzahl an erhaltenen/übermittelten Bytes der Periode innerhalb der Zeitspanne, in der das Netzwerkinterface am meisten das Netzwerk belastet hat.	Bytes
Übermittelte Bytes Maximum		
Erhaltene Bytes Durchschnitt	Durchschnittliche Anzahl an erhaltenen/übermittelten Bytes über die ganze Zeitspanne, in der gemessen wurde.	Bytes
Übermittelte Bytes Durchschnitt		

Tabelle 3.27.: Beschreibung der Werte vom Ergebnis einer Messung: Netzwerkbelastung pro Netzwerkinterface

3.3.3. Generierung von Datenverkehrströmen / Netzwerklast

In diesem Kapitel werden sämltiche Typen von Netzwerklast, die in dieser Arbeit vorkommen, beschrieben und Erzeugung erläutert.

Lasttyp	Ethernet	TCP	UDP
«MIN»: Besteht aus lauter Frames / Paketen, welche die minimal mögliche Grösse haben			
«RANDOM»: Beinhaltet Frames / Pakete, welche von zufälliger Grösse sind			
«MAX»: Setzt sich aus Frames / Paketen zusammen, die über die maximal mögliche Grösse verfügen.			

Tabelle 3.28.: 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 20 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 16) deaktiviert.
  - Bei den IP-Adressen handelt es sich um manuell zugewiesene Adressen.
- Sämtliche Applikationen 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.

3.3.4.1. UDP / Ethernet Vergleich

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.

Der Vergleich findet im PRP-Netzwerk statt, weil diese Umgebung im Fokus dieser Arbeit liegt. 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.

Für den Vergleich zwischen UDP und Ethernet wird auf dem DAN «srv01» jeweils die Netzwerklast mit dem «shck»-Skript wie folgt generiert, an «srv02» gesendet und parallel dazu mit tshark aufgezeichnet und mit «meas» gemessen:

1) Ethernet-Datenverkehr bestehend aus 10000 Frames mit minimaler Grösse:

```
1   sudo ./shck.py -d 00:0e:7f:30:17:4a -f ./testbin -i prp1 -n 10000 -s MIN -t ETH -P
```

2) Ethernet-Datenverkehr bestehend aus 10000 Frames mit maximaler Grösse:

```
1   sudo ./shck.py -d 00:0e:7f:30:17:4a -f ./testbin -i prp1 -n 10000 -s MAX -t ETH -P
```

3) Ethernet-Datenverkehr bestehend aus 10000 Frames mit zufälliger Grösse:

```
1   sudo ./shck.py -d 00:0e:7f:30:17:4a -f ./testbin -i prp1 -n 10000 -s RANDOM -t ETH -P
```

4) UDP-Datenverkehr bestehend aus 10000 Paketen mit minimaler Grösse:

```
1   sudo ./shck.py -d 192.168.0.2 -f ./testbin -i prp1 -n 10000 -s MIN -t UDP -P
```

5) UDP-Datenverkehr bestehend aus 10000 Paketen mit maximaler Grösse:

```
1   sudo ./shck.py -d 192.168.0.2 -f ./testbin -i prp1 -n 10000 -s MAX -t UDP -P
```

6) UDP-Datenverkehr bestehend aus 10000 Paketen mit zufälliger Grösse:

```
1   sudo ./shck.py -d 192.168.0.2 -f ./testbin -i prp1 -n 10000 -s RANDOM -t UDP -P
```

Darauf wird ein Vergleich zwischen den Netzwerkströmen 1 & 4, 2 & 5 und 3 & 6 erstellt.

3.3.4.2. Szenario 01: Performance im PRP-Netzwerk

In diesem Szenario wird die ursprüngliche Konfiguration der Testumgebung verwendet, um die Ergebnisse dieses Szenarios als Referenzpunkt für weitere Messungen verwenden zu können. Damit dieses Szenario als Referenz betrachtet werden kann, ist es notwendig, dass alle in dieser Arbeit generierten Datenstöme hier untersucht werden.

3.3.4.3. Szenario 02: Performance ohne PRP

3.3.4.4. Szenario 03: Effekt von Laufzeitunterscheidungen zwischen Netzwerk A und B

3.3.4.5. Szenario 04: Zeitweiser Ausfall eines Netzwerkpfades

3.3.4.6. Szenario 05: Abhängigkeit vom verwendeten Protokoll

TCP, UDP, Ethernet

3.3.4.7. Szenario 06: Einfluss nicht entfernter Duplikate auf Funktion und Performance

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

3.3.4.9. Szenario 08: Einluss von Offload-Mechanismen

4. Resultate und Interpretation

4.1. Szenario 01: Performance im PRP-Netzwerk

Es folgt ein ausgezeichnetes Histogramm, generiert aus einer Textdatei:

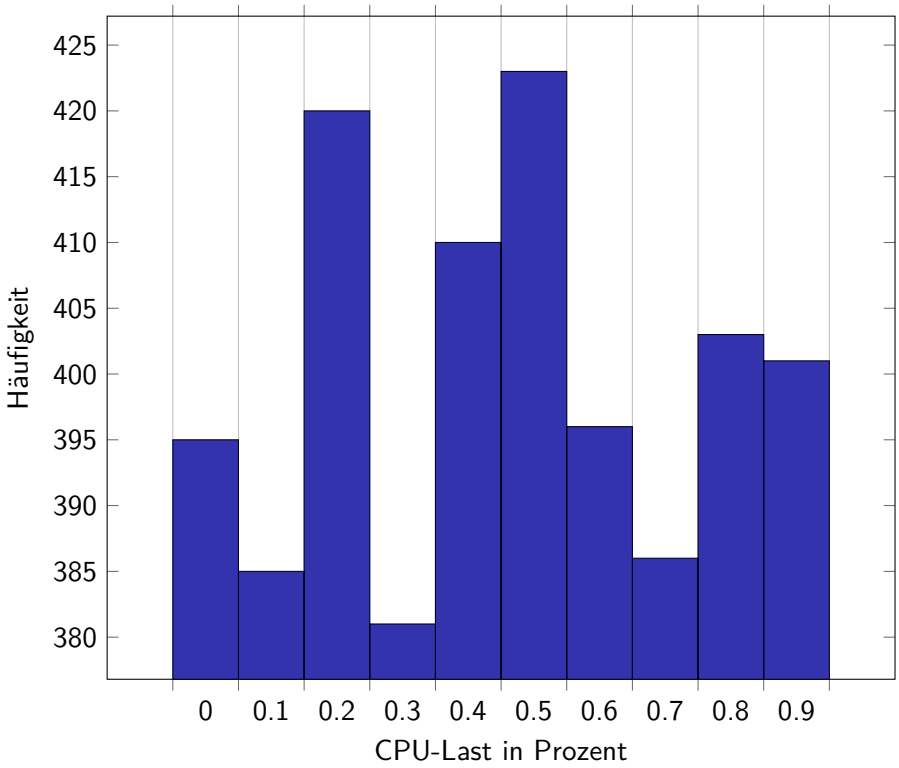

Abbildung 4.1.: Test Histogram

4.2. Szenario 02: Performance ohne PRP

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

4.4. Szenario 04: Zeitweiser Ausfall eines Netzwerkpfades

4.5. Szenario 05: Abhängigkeit vom verwendeten Protokoll

TCP, UDP, Ethernet

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

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

4.8. Szenario 08: Einluss von Offload-Mechanismen

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. Cahalan: *pmap(1) - Linux man page @http://linux.die.net/man/1/pmap*.

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

[3] 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*.

[4] A. Engelen: *NetHogs @http://nethogs.sourceforge.net*.

[5] A. Gemperli: *Bedienungsanleitung Impairment Generator (ECI-IG)*.

[6] A. Zimmerman et al.: *flowgrind @http://www.flowgrind.net*.

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

[8] ESnet / Lawrence Berkeley National Laboratory: *iperf3 @http://software.es.net/iperf*.

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

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

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

[12] G. Combs et al.: *Offloading @https://wiki.wireshark.org/CaptureSetup/Offloading*.

[13] G. Langeveld: *atop @http://www.atoptool.nl/index.php*.

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

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

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

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

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

[19] 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*.

[20] 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*.

[21] 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*.

[22] J. Corbet: *Generic recieve offload @https://lwn.net/Articles/358910/*.

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

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

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

[26] J. Seward et al.: *Valgrind @http://valgrind.org/*.

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

[28] L. Deri: *ntop @http://www.ntop.org/*.

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

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

[31] M. Brown: *arping @http://linux-ip.net/html/tools-arping.html*.

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

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

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

[35] P. Biondi: *Scapy @http://www.secdev.org/projects/scapy*.

[36] R. Jones: *netperf @http://www.netperf.org/netperf/*.

[37] S. Godard: *sysstat @http://sebastien.godard.pagesperso-orange.fr*.

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

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

[40] TCPDUMP-Team: *TCPDUMP/LIBPCAP public repository @http://www.tcpdump.org*.

[41] 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.

[42] Wikipedia: *Large segment offload @http://en.wikipedia.org/wiki/Large_segment_offload*.

7. Glossar

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.

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.

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.

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

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.

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.

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.

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

TX **Transmitter** / Sender.

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 [16] 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 [5] 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. [10]

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-linu` nach `/usr/lib/jvm/java-7-openjdk/jre/lib/amd64`.

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