h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt
- Fachbereich Informatik -

Bestimmung der optimalen Paketgröße in einem Netzwerk mittels eines Frameworks um Zabbix

Abschlussarbeit zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

vorgelegt von

Can Kedik
731620

Referent:	Prof. Dr. Ronald C. Moore
Korreferent:	Prof. Dr. Bettina Harriehausen-Mühlbauer

Abstract

Die vorliegende Arbeit beschäftigt sich mit der Analyse eines Local Area Netzwerks bestehend aus Raspberry Pis. Die zentrale Fragestellung dieser Arbeit ist, ob es eine für gegebenes, homogenes Netzwerk eine optimale Dateigröße zur Übertragung im Netzwerk gibt. Dafür wird als erstes ein Einblick in die verwendete Hardware und den Aufbau des Netzwerkes gegeben, und das Framework, das zum Testen des Netzwerkes erstellt wurde erläutert. Dabei werden weitere Einblicke in das Open Source Netzwerk-Monitoringsystem Zabbix gegeben. Danach werden die gesammelten Daten zu mehreren Testfällen analysiert und nach zuvor festgelegten Kriterien bewertet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Einführung in das Thema

Computernetzwerke sind inzwischen fester Bestandteil der Gesellschaft und finden überall Verwendung. Wenn früher Netzwerke nur im Militär, in Universitäten und Firmen benutzt wurden, haben nun auch normale Endverbraucher Zugriff auf Netzwerke. Jedoch ist auch das Internet nur ein Netzwerk aus Netzwerken, das über den gesamten Globus verteilt ist. Geht man in die unterste Ebene, kommt man irgendwann in einem sogenannten Local Area Network (LAN) an. Inzwischen sind diese in fast jedem Haushalt zu finden, da sie es ermöglichen mehrere Computer über einen Router mit dem Internet zu verbinden. LANs ermöglichen einen Datenaustausch zwischen den im Netzwerk verbundenen Computern. Ein sehr beliebter Verwendungszweck sind die sogenannten LAN-Partys, bei denen über ein LAN Videospiele miteinander gespielt werden können. Dabei ist es möglich lokale verteilte Systeme aufzubauen, die gemeinsam eine Aufgabe bearbeiten. Das Berkeley Open Infrastructure for Network Computing stellt eine Anwendung für verteilte Systeme dar [Cal16]. Mit dieser ist es möglich über das Internet an verschiedenen Forschungsprojekten teilzunehmen, indem man eigene Rechenzeit zur Verfügung stellt. Eine weitere Anwendung für verteilte Systeme stellt das Zabbix Framework dar, welches auch in dieser Bachelor Arbeit näher betrachtet wird. Zabbix ist ein Netzwerk-Monitoring Tool, mit dem es möglich ist Störungen in einem Netzwerk zu erkennen und die Leistung von den im Netzwerk angeschlossenen Computern zu betrachten.

1.2 Motivation und Zielsetzung

In einem Netzwerk, in dem konstant Daten ausgetauscht werden, wirkt sich die Größe der versandten Pakete auf die Leistung der Endgeräte aus. Um eine gleichmäßige Auslastung des Netzwerkes und die Leistungsfähigkeit der im Netzwerk angeschlossenen Computer zu gewährleisten, kann man dies mittels der Größe der Pakete steuern. Deshalb ist das Bestimmen einer Paketgröße für ein gegebenes System wichtig, um eine gleichmäßige Lastenverteilung im Netzwerk zu haben und die Endgeräte nicht zum Abstürzen zu bringen. Das Ziel dieser Bachelorarbeit ist es herauszufinden, ob über die Paketgröße eine Optimierung der Performanz der Endgeräte und der Datenübertragung im Netzwerk möglich ist. Dabei wird versucht die größtmögliche Menge an Daten im Netzwerk auszutauschen ohne die Funktionalität eines

der Hosts zu gefährden.

1.3 Methoden zum Erreichen des Ziels

Zabbix, welches von der Zabbix SIA vertrieben wird, ist wie Nagios eine Open Source Netzwerkmonitoring Software, die es ermöglicht Geräte in einem Netzwerk zu überwachen. Dafür wurde ein Local Area Network bestehend aus vier Raspberry Pis aufgebaut. Das Netzwerk wurde in einer Sterntopologie aufgebaut, welche standardmäßig verwendet wird. Der Vorteil bei dieser Topologie ist, dass der Ausfall von einem Computer keine Auswirkung auf den Rest des Netzwerks hat, sie leicht erweiterbar ist und hohe Übertragungsraten bietet, wenn der Netzknoten ein Switch ist [Tan09]. Dafür wird das auf dem TCP/IP-Protokollstapel basierende Secure Copy Programm verwendet. Es stellt eine Erweiterung der Unix Secure Shell dar und ermöglicht einen fehlerfreien Austausch von Dateien. Beide Programme, Secure Copy und Secure Shell sind, inzwischen fester Bestandteil fast aller Linux Distributionen, so auch dem Debian Port Raspbian, welcher auf den Raspberry Pis installiert ist. Mit diesen Tools wurde im Rahmen dieser Bachelorarbeit ein eigenes Testframework aufgebaut, welches automatisiert Dateien von den Computern verschickt und Informationen über die Dateien,

Abbildung 1.1: ISO/OSI Referenzmodell.
Das Framework befindet sich im ISO/OSI Referenzmodell auf der Anwendungsschicht.

Empfänger und Übermittlungszeiten speichert. Diese Logfiles werden mittels Textverarbeitung analysiert und in Tabellen dargestellt. Zabbix kann Statistiken zu den Computern im Netz erstellen. So ist es möglich mittels Zabbix die CPU-Last, die Festplattenauslastung und die Last auf den Ethernet Ports der Raspberry Pis zu beobachten. Dieses Framework wird in Kapitel 2 nochmal genauer vorgestellt. Außerdem werden in dieser Arbeit drei verschiedene Testfälle betrachtet, kleine Dateien 20 Megabyte, mittlere Dateien 200 Megabyte und Große Dateien mit 2000 Megabyte. Zum Schluss werden die zuvor gesammelten Ergebnisse in Kapitel 5 verglichen und die Annahme überprüft, ob es möglich ist eine optimale größe für Pakete in einem Netzwerk zu bestimmen.

2 Das Framework

2.1 Verwendete Hardware

In diesem Framework wird folgende Hardware verwendet:

- Zwei Switches,
- Vier Raspberry Pis der ersten Generation,
- Ein Raspberry Pi der zweiten Generation,
- Mehrere Ethernet Kabel.

Diese Geräte wurden in einem eigenständigen Netzwerk zusammengeschaltet. So sind an jedem Switch zwei Pis der ersten Generation angeschlossen während an einem der Switches der Pi der zweiten Generation angeschlossen, ist (siehe Abb. 2.1). Welche Software auf den Pis verwendet wurde, wird in Abschnitt 2.3 erklärt. In der Tabelle 2.1 kann sind die Spezifikationen der Raspberry Pis angegeben. Die beiden verwendeten Switches arbeiten mit 100 oder 1000 Mbit/s, doch aufgrund der verwendeten Ethernet Ports auf den Raspberry Pis, welche eine Übertragungsrate von 100 Mbit/s haben, ist der Austausch der Pakete zwischen den Pis auf 100 Mbit/s beschränkt und kann so das ganze Potential der Switches nicht ausnutzen.

2.2 Raspberry Pi

Bevor es in den Aufbau des Frameworks geht, wird noch ein Einblick in die Raspberry Pis gegeben. Der Raspberry Pi ist eine Entwicklung aus England. Eben Upton, der 2006 Dozent an der Cambridge University war, stellte zu der Zeit fest, dass der Informatik Wissenstand von neuen Studenten in Cambridge im Bereich der Informatik sehr gering ist.

Gerät	CPU MHz	Arbeitspeicher MB	Speicher GB	Netzwerk Port MB/s
Raspberry Pi 1	700 MHz	512 MB	8 GB	100 MB/s
Raspberry Pi 2	900 MHz	1024 MB	32 GB	100 MB/s

Tabelle 2.1: Spezifikation der Raspberry Pis

In einem Interview mit dem Online Magazin *readwrite* sagte Upton[Ors14]

> "We'd kind of pulled the ladder up after us. We built these very sophisticated and user-friendly computers for children to use now. Or not even computers – game consoles and phones and tablets, kind of appliances. But people were being denied that opportunity to tinker. So really Raspberry Pi is an attempt to get back – without kind of being too retro – some of what we kind of feel was lost from the evolution of computers over the last 25 years."

Upton, der nach eigener Aussage während seiner Schulzeit mit einem BBC-Micro Computer erste Erfahrungen im Programmieren machte, betont auch, dass der Raspberry Pi dafür gedacht ist, Schülern einen Computer zu geben, der erschwinglich und leicht modifizierbar ist. Deshalb ist es Upton auch wichtig, dass die Pis keine hohen Kosten haben. Mit einem Preis von 40 Euro für den Raspberry Pi 1 [Ama16a], 36,65 Euro für den Raspberry Pi 2 Modell B [Ama16b] und 42,70 Euro für den am Anfang des Jahres 2016 veröffentlichten Raspberry Pi 3 Modell B [Ama16c] ist dies auch gelungen. Aufgrund des geringen Preises eignen sich Raspberry Pis ausgezeichnet dafür Client Server Anwendungen mit realen Geräten aufzubauen. Deshalb werden Raspberry Pis zum Überprüfen der These die, in dieser Arbeit behandelt wird, verwendet.

2.2.1 Entwicklung der Pis

Der erste Raspberry Pi wurde im Jahr 2012 veröffentlicht. Jedoch gab es schon 2006 einen ersten Prototypen. Dieser verwendete einen Atmel-ATmega644-Mikrocontroller, welcher allerding nicht leistungsfähig genug war. Da zu dieser Zeit der Boom der Smartphones anfing, kamen aber auch immer mehr ARM-Prozessoren auf den Markt. So entschied man sich dafür den Broadcom BCM2835 Prozessor mit einer ARMv6 Architektur zu verwenden. Dieser ist leistungsfähig genug um Spiele wie Quake 3 Arena und H.264 komprimierte Videos abzuspielen. Seit Herbst 2012 ist der Raspberry Pi Model B im Handel. Bei diesem wurde der Arbeitsspeicher auf 512 Megabyte erhöht. Es ist auch das Modell welches für die Hosts in diesem Framework verwendet wurde. Am 14. Juli 2014 wurde das Modell B+ vorgestellt, mit welchem auch eine offizielle Spezifikation für Erweiterungsplatinen veröffentlicht wurde. Das ermöglicht es, den Raspberry Pi nach Belieben zu erweitern. Vier Monate später am 14. November 2014, wurde das Modell A+ vorgestellt. Dieses zeichnet sich dadurch aus, dass die Platine kleiner ist als die der Vorgängermodelle und auch weniger kostet. Der Raspberry Pi 2 Modell B wurde am 2. Februar 2015 vorgestellt. Dieser wird in diesem Framework als Zabbix Server verwendet wird. Der Raspberry Pi 2 verwendet einen Vierkern Broadcom BCM2836 Prozessor und ist der erste Raspberry Pi, der eine ARMv7 Architektur verwendet. Mit 900 MHz hat dieser einen 200 MHz schnelleren Prozessor als die Vorgängermodelle. Auch der Arbeitsspeicher wurde aufgerüstet. Er wurde verdoppelt und hat nun einen 1024 Megabyte großen Arbeitsspeicher. Auf der Entwicklerkonferenz Build 2015 wurde von Mi-

crosoft Windows 10 IoT angekündigt, was ein Windows 10 Port ist, welcher für *Internet of Things*-Geräte entwickelt wurde, der vom Raspberry Pi 2 unterstützt wird [der15]. Am 26. November 2015 wurde der Raspberry Pi Zero präsentiert, dieser hat eine ähnliche Ausstattung wie der Raspberry Pi 1 Modell B, hat jedoch einen auf 1 GHz getakteten Prozessor. Das neueste Modell der Raspberry Pi Familie (Stand Juni 2016) ist der Raspberry Pi 3 Modell B. Dieser ist der erste Raspberry Pi, der eine 64bit-ARMv8 Architektur verwendet und mit dem Broadcom BCM2837 Prozessor ausgestattet ist. Dieser Prozessor ist auf 1,2 GHz getaktet. Dies ist auch der erste Pi, welcher einen Intergriertes W-Lan und Bluetooth Low Energy hat. Am Arbeitsspeicher dieses Pis wurde nichts verändert, so besitzt auch dieser wieder 1024 Megabyte Arbeitsspeicher LPDDR2-SDRAM.

2.3 Aufbau der Software

Das Testframework besteht aus drei verschiedenen Teilen.

- Zabbix
- Eigenentwicklung auf dem Server
- Eigenentwicklung auf dem Agent

Die Eigenentwicklungen sind alle mit Bash Skript programmiert, Zabbix hingegen ist eine bereits fertige Open Source Lösung Im folgenden Abschnitt wird ein Einblick in diese beiden Komponenten gegeben.

2.3.1 Zabbix

Zabbix ist ein Open Source Netzwerk-Monitoringsystem. Die erste Version wurde von Alexei Vladishev entwickelt [SIA16a]. Ein weiterer bekannter Vertreter der Netzwerk-Monitoringsysteme ist Nagios [LLC16], welches wie Zabbix unter der GPL vertrieben wird, womit jedem frei steht Zabbix zu verändern und zu erweitern. Beide Systeme basieren auf einer Client-Server Architektur. Im Weiteren wird jedoch nur Zabbix betrachtet, welches aus zwei Komponenten besteht.

Zabbix Server Der Server hat eine auf PHP basierende Weboberfläche über die es für den Benutzer möglich ist die Agents zu konfigurieren. So können die Templates manuell erstellt werden, die den Zabbix Agents mitteilen, welche Informationen dem Server zu übermitteln sind. Einen genaueren Einblick in den Zabbix Server gibt es in Abschnitt 3.1.

Zabbix Agent Die Clients, die im Netzwerk überwacht werden sollen, sind die sogenannten Agents. Sie leiten Informationen an den Server weiter, die von diesem gefordert werden. In den späteren Kapiteln wird der Hauptfokus auf der Auslastung der Festplatte, der

CPU und des Ethernet Ports liegen. Einen genaueren Einblick in den Zabbix Agent gibt es in Abschnitt 3.2.

2.3.2 Eigenentwicklung

Die selbstentwickelte Software wird in zwei Kategorien unterteilt. Ein Teil der Software läuft auf den Agents. Diese haben den Zweck Netzwerk Traffic zu erzeugen. Dadurch entsteht auf dem Netzwerk und auf den Zabbix Agents eine Nutzlast, die vom Zabbix Server gesammelt werden kann. Der zweite Teil der Software läuft auf dem Server. Die Software auf dem Server unterstützt den Entwicklungsprozess auf den Agents.

1. Zabbix Server

   Update Script: Dieses Script wird von der Software Entwicklungsmaschine ausgeführt. Es aktualisiert den Code auf den Endgeräten, die die Programme Hintergrundrauschen, Pinger, Synchronize und Startrauschen aktualisieren. Dabei wird mittels Secure Copy der Quellcode auf das Endgerät gespielt.

   Get logs: Die Skripte Pinger und Hintergrundrauschen erstellen jeweils auf den Endgeräten Logfiles. Da es jedoch ein sehr hoher Verwaltungsaufwand wäre auf den Endgeräten die Logfiles weiterzuverwerten, werden die auf den Agents gelagerten Logfiles mit dem Skript Get Logs auf dem Rechner gesammelt, der dieses startet. Somit hat man die von den Endgeräten gesammelten Logfiles auf einem Rechner und kann mit der Weiterverarbeitung der Logfiles beginnen.

2. Zabbix Agent

   Hintergrundrauschen: Diese Eigenentwicklung stellt mit Zabbix die Kernkomponente des Frameworks dar. Wie der Rest der selbstentwickelten Software, wurde sie komplett in Bash programmiert, da das Framework ausschließlich in einer Linux Umgebung entwickelt, getestet und verwendet wird. Hintergrundrauschen schickt über Secure Copy Pakete von einem Agent zum anderen. Secure Copy baut auf dem SSH Protokoll auf [Bor]. So wird eine TCP Verbindung zum angesprochenen Host aufgebaut und eine Last auf dem Netzwerk und den Endegeräten erzeugt, die mit Hilfe des Zabbix Servers gemessen werden kann. Außerdem speichert Hintergrundrauschen die Dauer, die ein Paket benötigt, um erfolgreich bei seinem zufällig ausgewählten Empfänger anzukommen. Es werden drei verschieden große Pakete verschickt: 20 Megabyte, 200 Megabyte und 2 Gigabyte. Die Ergebnisse der Logfiles werden in Kapitel 4 betrachetet.

   Startrauschen: Startrauschen wird automatisch auf den Endgeräten ausgeführt. Es dient als eine Zeitschaltuhr und ermöglicht ein zeitversetztes Starten des Scripts Hintergrundrauschen. Jedoch wird in den in dieser Ausarbeitung betrachteten

Tests immer ein zeitgleicher Start durchgeführt. Trotzdem wird dieses Skript weiterhin verwendet, da es eine einfache Möglichkeit darstellt, die Tests zu erweitern.

Synchronize: Raspberry Pis besitzen keine eigene Batterie wie sie in handelsüblichen Rechnern verbaut werden. Deshalb ist nach jedem Neustart die Uhrzeit der Pis unzuverlässig. Dieses Programm synchronisiert die Uhrzeiten der Agents mit der Zeit des Zabbix Servers, welcher zwischen den Tests nicht neu gestartet wird. Sychnronize baut eine Verbindung mit dem Pi DotA auf und fragt von diesem die Uhrzeit ab. Dies geschieht über eine Secure Shell Verbindung zum Zabbix Server

```
TIMESERVER=192.168.2.116
DATE=‘sshpass -p ’raspberry’ ssh 192.168.2.116 "date +%s"‘
sudo date  -s @$DATE
```

Pinger: Pinger wird zusammen mit dem Hintergrundrauschen ausgeführt und ist um den Linux eigenen Pingbefehl herum aufgebaut. Mittels Pinger wird die Latenz unter den einzelnen Endgeräten gemessen.

```
Ping 192.168.2.250 | while read pong;
do echo "[$(date)] $pong";
done >> ~/logfiles/ping/TinkerPing20MB &
```

Wie man sieht, wird als Erstes der Pingbefehl ausgeführt. Über die Pipe wird dieser jedoch weitergeleitet und in einer Schleife weiterverarbeitet. Neben der Meldung, die vom Befehl Ping kommt, wird noch ein Datum vorgestellt. Diese ganze Meldung wird dann in einer Logfile Datei abgespeichert.

2.4 Einsatz des Frameworks

Mit dem Einsatz, der im vorherigen Abschnitt vorgestellten Software, ist das Testframework aufgebaut. In Abb. 2.1 wird dargestellt, wie die einzelnen Komponenten zusammenspielen. Hier sieht man, dass an einem Switch zwei Raspberry Pis und an einem anderen Switch drei Raspberry Pis angeschlossen sind. Einer von diesen Pis ist der Zabbix Server, der die aktiven Hosts im Netzwerk überwacht. Um das Framework zu starten, muss man als Erstes den Zabbix Server DotA starten. Wenn dieser mit dem Bootivorgang abgeschlossen hat, kann man die restlichen Raspberry Pis einschalten. In den Agents wird nun als Erstes das Programm Synchronize ausgeführt. Da Raspberry Pis keine eigene Uhr haben, aber die Logfiles und der Zabbix Agent von der Zeit abhängig sind um korrekt arbeiten zu können, müssen die Uhren synchronisiert werden. Das Skript Synchronize wird aus der Autostart Konfigurationsdatei von den Raspberry Pis ausgeführt. Deshalb gibt es auch keine Probleme die Uhrzeit zu setzen, da dieses Programm als Super User ausgeführt wird. Nachdem dieses Programm erfolgreich ausgeführt wurde, startet das Startrauschen Skript. Dieses Programm ermöglicht einen zeitversetzten Start von Hintergrundrauschen und Pinger. In Hintergrund-

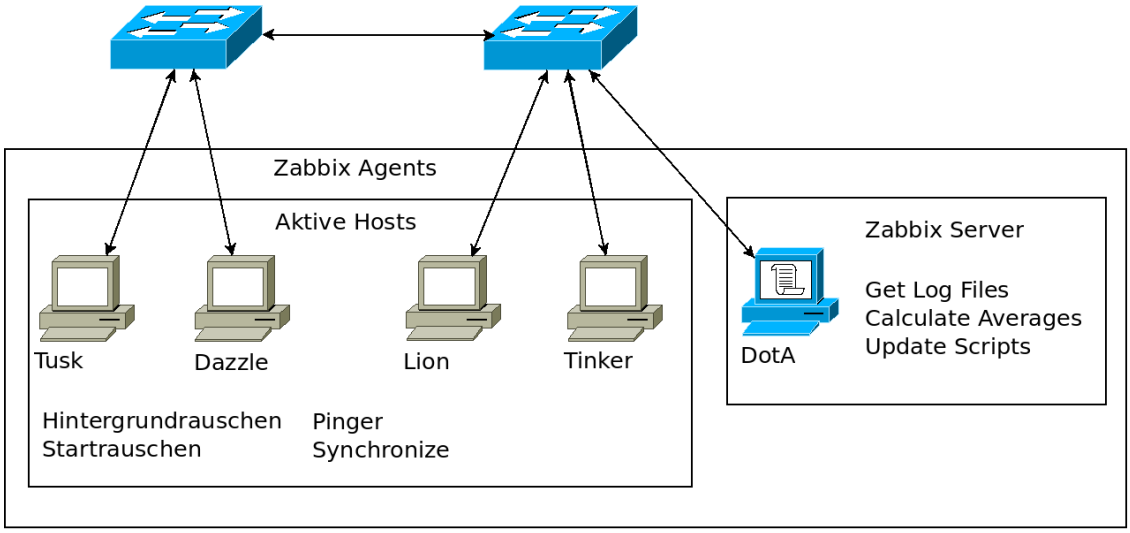

Abbildung 2.1: Aufbau des Netzwerks.

rauschen müssen jedoch immer kleine Veränderungen vorgenommen werden. So muss je nach durchzuführendem Test folgende Zeile umgeschrieben werden.

```
FILESIZE=500
```

Diese Variablendeklaration muss immer dem auszuführenden Test angepasst werden. Da die Datei, die verschickt wird, über den Linux internen Befehl *duplicate data* geschrieben wird, muss man diesem eine Blockgröße (bs=) und eine Anzahl an zu schreibenden Blöcken (count=) mitgeben.

```
dd if=/dev/urandom of=$MYRANDOMFILE bs=4M count=$FILESIZE
```

Die Blockgröße beträgt immer 4 MB, über die Variable SIZE kann man die Anzahl der Blöcke bestimmen. Würde man zum Beispiel SIZE=10 setzen, würde der *duplicate data* Befehl einen 40 Megabyte großen Block erzeugen. Im Fall der in dieser Ausarbeitung betrachteten Testfälle wurde die SIZE 5, 50 und 500 gewählt, welche dann eine 20 Megabyte, 200 Megabyte und 2000 Megabyte große Datei erzeugen. Ist dieser Prozess abgeschlossen, beginnt das Framework zu arbeiten. Die Daten werden zwischen den Agents verschickt und man kann mit der Auswertung beginnen.

3 Zabbix

In Kapitel 2 wurde schon die verwendete Netzwerk-Monitoring Software angeschnitten. In diesem Abschnitt wird nun ein tieferer Einblick in den Aufbau und die Verwendung von Zabbix gewährt. Zabbix ermöglicht es auch das Monitoring als ein verteiltes System mit mehreren Servern aufzubauen. Diese sogenannten Proxys sind jedoch kein Bestandteil dieser Thesis und werden deshalb nicht weiter betrachtet.

3.1 Zabbix Server

Der Zabbix Server ist die zentrale Komponente des vorgestellten Frameworks. Dem Server werden von den Agents und Proxys Informationen zugeschickt. Die Aufgabe des Servers ist es, die Daten zu speichern und zu verarbeiten. Der Server speichert auch die Konfiguration der einzelnen Agents, die sich im Netzwerk befinden. Um die Konfiguration der Agents einstellen zu können, müssen jedoch erst einmal die Agents im Zabbix Server registriert werden. Die dazu benötigten Daten sind die Agent IP und ein eindeutiger Name. Wenn nun der Agent im Server registriert worden ist, kann man ihn im Zabbix Server verschiedene Templates auflegen. Das in Kapitel 2 gezeigte Framework verwendet primär das Template *Template OS Linux*. Dieses Template ist eines von vielen vordefinierten Templates. Es ist auch möglich selber Templates zu erstellen. Diese Templates sind in einem XML-Format abgespeichert und können dadurch einfach unter Nutzern von Zabbix ausgetauscht werden. Die Firma, die den Vertrieb von Zabbix regelt, hat extra dafür die Zabbix Share eingeführt auf dem Zabbix Templates und vieles mehr ausgetauscht werden kann [SIA16d].
Templates enthalten sogenannte Items, die über Active Checks oder Passive Checks Informationen von einem vorher im Zabbix Server registrierten Agent anfordern. Ein Passive Check geschieht über einen Request. Diese sind in JSON verfasst und werden gebündelt verschickt um Bandbreite zu sparen. Der Prozess beeinhaltet fünf Schritte [SIA16c]:

1. Der Server öffnet eine TCP Verbindung.
2. Der Server sendet einen Request. Als Beispiel agent.version.
3. Der Agent empfängt den Request und antwortet mit der Versionsnummer.
4. Der Server verarbeitet die Antwort und erhält als Ergebnis Zabbix3.0.0rc.
5. Die TCP Verbindung wird geschlossen.

Passive Checks und Active Checks unterscheiden sich darin, wer den Request auslöst. Bei einem Passiv Check sind die Hosts inaktiv bis vom Server ein Request eintrifft. Bei Active Checks wird der Agent selber aktiv. Der Agent sendet dann einen Request an den Server, in dem der Agent nach den Daten fragt, die der Server erwartet [Kra16]. Dies ist der Fall, wenn vom Agent Daten abgefragt werden sollen, die nicht vom Betriebssystem des Agents gesammelt werden. Das bedeutet, dass der Agent selber nun die Daten, die der Server erwartet, sammeln muss. Dies geschieht meistens über externe Programme. Sollte ein Agent zu viele Active Checks haben, kann es sich auf die Performanz des Hosts auswirken.

Über die API ist es möglich Programme für den Server zu schreiben, mit denen man zum Beispiel eine eigene Benutzeroberfläche erstellen kann um die Konfigurationen auf dem Server zu verwalten. Über die API ist es auch möglich einen eigenen Zugriff auf die dem Server zugrunde liegende Datenbank herzustellen. Der Zabbix Server basiert auf einem Apache Web Server. Um das Zabbix Frontend nutzen zu können, wird PHP 5.4.0 oder aufwärts benötigt. PHP 7 wird jedoch noch nicht unterstützt. Die Daten und Statistiken, die Zabbix sammelt, werden in einer Datenbank gespeichert. Es werden fünf verschiedene Datenbanken unterstützt die im folgenden genannt werden [SIA16b]:

- MySql Version 5.0.3 aufwärts.
- Oracle Version 10g aufwärts.
- PostgreSQL Version 8.1 aufwärts.
- SQLite Version 3.3.5 aufwärts.
- IBM DB2 Version 9.7 aufwärts (Noch nicht fehlerfrei).

Der Zabbix Server selber ist auch als ein Agent konfiguriert, der sich selber überwacht. Der Server ist jedoch nicht im allgemeinen Prozess des in Kapitel 2 vorgestellten Frameworks tätig. Der Server gibt Aufschluss über die sogenannte *Zabbix Server Performance*. Diese betrachtet zwei Dinge, einmal die sogenannte Queue, welche angibt wieviele von den von den Agents übermittelten Werten darauf warten vom Zabbix Server verarbeitet zu werden, und die verarbeiteten Werte pro Sekunde. Diese Kennzahlen lassen einen Schluss auf die Leistungsanforderungen des Servers zu. Sollte die Queue einen bestimmten Schwellenwert erreichen, wird auf dem Zabbix Frontend eine Warnung ausgegeben, dass der Server mit der Verarbeitung nicht hinterherkommt. Dies kann so weit gehen, dass die gesammelten Daten auf dem Server fehlerhaft sind. Jedoch ist dieses Szenario in diesem Framework unwahrscheinlich, da die Größe des Frameworks überschaubar ist. Der Traffic auf dem Ethernet Port Eth0 wird überwacht, da alle von den Agents gesammelten Informationen über diesen an den Server gelangen. Dabei wird der eingehende sowie der ausgehende Traffic betrachtet.

3.2 Zabbix Agent

Der Zabbix Agent wird auf dem zu überwachenden System installiert. Der Agent sammelt die Daten über die im Betriebssystem integrierte Monitoring Funktion oder über die Zabbix eigene API und sendet diese Informationen je nach Art des Checks an den Agent. Da der Agent schon im Betriebssystem integrierte Funktionen benutzt ist, dieser sehr effizient und verbraucht kaum Ressourcen des Host Systems. Anders als der Server, besitzt der Agent kein eigenes Frontend und speichert die Werte nicht in einer Datenbank. Der Agent ist so konstruiert, dass dieser weitestgehend ohne Administratorrechte funktionieren kann. Da es auch möglich ist Agent Hostsysteme über einen Fernzugriff auszuschalten oder neuzustarten, müssen dem Agent für diese Funktionen die Rechte zugeteilt werden. Dadurch wird das Sicherheitsrisiko für den Host erheblich reduziert. Für gewöhnlich wird für den Agent ein eigener User angelegt. Die Anwendung des Agents läuft unter Unix Systemen als ein Daemon und unter Windows als ein Systemdienst. Zabbix unterstützt jedoch noch mehr Betriebssysteme. Eine Auswahl davon sind [SIA]:

- Linux.
- Windows: alle Desktop und Server Versionen seit 2000.
- OpenBSD.
- Mac OS X.
- Solaris 9,10,11.

Im Aufbau des Frameworks, das in dieser Arbeit weiter betrachtet wird, laufen die Zabbix Agents unter dem gängigen Raspberry Pi Betriebssystem Raspbian, welches ein Debian Port ist. Um den Host jedoch verwenden zu können, muss man in den Konfigurationsdateien die Ports, die der Server für das Versenden von Requests benutzt, muss man einstellen. Auch die IP des Servers muss eingetragen werden, da der Agent sonst eingehende Requests verwirft um so die Sicherheit für den Host zu gewährleisten. In den Konfigurationsdateien der Agents ist es auch möglich die Active Checks zu notieren, was in diesem Versuchsaufbau auch gemacht wurde. Das Template OS Linux hatte keine Items, die das Überwachen von der Festplattenauslastung gewährleisten. Dies kann man über die Konfigurationsdateien einstellen:

```
UserParameter=custom.vfs.dev.read.ops[*],customParaScript.sh $1 4
```

Wenn dann in der Antwort zu einem Active Check nach dem *custom.vfs.dev.read.ops[*]* gefragt wird, wird das Bash Script customParaScript.sh ausgeführt. Die Auszeichnung *UserParameter=* gibt an, dass es sich hierbei um ein selbstdefiniertes Item handelt, welches mithilfe von Zabbix überwacht werden soll. Das Script customParaScript.sh dient dazu die Last auf der Festplatte zu überprüfen. Dazu liest es aus der Datei die in Unix Systemen

unter /proc/diskstats liegt. Die zugehörigen Werten $1 und 4 sind Variablen, die für die Verarbeitung notwendig sind. Der in diesem Beispiel genannte User Parameter liest die Leseoperationen pro Sekunde aus der Datei und leitet diese an den Server weiter.

4 Versuche

Aufgrund der unterschiedlichen Paketgrößen wird das Hostsystem unterschiedlich belastet. Mit den folgenden drei Versuchen soll die optimale Paketgröße gefunden werden. Dazu werden die vom Zabbix Server und der selbstentwickelten Software zur Verfügung gestellten Daten mit Mitteln der Stochastik analysiert. Es werden der Durchschnitt $\varnothing$ und die Standardabweichung σ der folgenden Werte betrachtet.

- Auslastung auf dem Ethernet Port
- Festplattenauslastung
- CPU-Auslastung
- Anzahl der erfolgreich verschickten Pakete
- Menge der versendeten Byte

Die Ergebnisse aus Abschnitt 4.2, Abschnitt 4.3 und Abschnitt 4.4 werden in Kapitel 5 miteinander verglichen.

4.1 System im Ruhezustand

Bevor mit der Auswertung der Testergebnisse in Abschnitt 4.2 bis Abschnitt 4.4 begonnen wird. Soll das gesamt Framework erstmal in einen Ruhestand gezeigt werden. Das soll ermöglichen eine bessere Abschätzung der einzelnen Werte in den folgenden Testfällen zu erhalten. In diesem Test wird keines der in Abschnitt 2.3.2 selbstentwickelten Programme ausgeführt. Die einzige Software die auf den Systemen läuft ist sind die Zabbix Agents und der Zabbix Server, welcher nötig ist um die Daten zu sammeln.

Dazu wird als erstes der eingehende und der ausgehende Traffic auf den Ethernet Ports betrachtet. Wie man sehen kann sind die Ethernet Ports kaum ausgelastet mit einem durchschnittlich eingehenden Datenstrom von 2,20 Kilobit/s. Auch der maximal ausgehende Datenstrom ist sehr gering mit einem Durchschnitt von 2,98 Kilobit/s. Da der durchschnittliche Traffic näher am minimalen Traffic liegt sieht man, dass die Hosts fast keinen eingehenden Traffic zu bearbeiten haben. Der Traffic wird von Zabbix selber erzeugt, der in regelmäßigen Abständen nach dem Zustand der Agents abfragt. Dies ist auch der Grund für den ausgehenden Traffic. Dieser ist in diesem Test generell höher als der eingehende Traffic, da die

Agent	Minimal Kb/s	Durchschnitt Kb/s	Maximal Kb/s
Dazzle	2,08 Kb/s	2,19 Kb/s	2,99 Kb/s
Tusk	2,06 Kb/s	2,23 Kb/s	3,12 Kb/s
Tinker	2,07 Kb/s	2,19 Kb/s	2,90 Kb/s
Lion	2,04 Kb/s	2,19 Kb/s	2,91 Kb/s
Agent ∅	2,06 Kb/s	2,20 Kb/s	2,98 Kb/s
Agents σ	0,01 Kb/s	0,02 Kb/s	0,09 Kb/s

Tabelle 4.1: Eingehender Traffic auf den Ethernet Ports im Ruhezustand auf allen Pis.

Agent	Minimal Kb/s	Durchschnitt Kb/s	Maximal Kb/s
Dazzle	2,57 Kb/s	2,72 Kb/s	3,31 Kb/s
Tusk	2,54 Kb/s	2,67 Kb/s	3,62 Kb/s
Tinker	2,58 Kb/s	2,72 Kb/s	3,46 Kb/s
Lion	2,49 Kb/s	2,72 Kb/s	3,26 Kb/s
Agent ∅	2,55 Kb/s	2,71 Kb/s	3,41 Kb/s
Agents σ	0,04 Kb/s	0,02 Kb/s	0,14 Kb/s

Tabelle 4.2: Ausgehender Traffic auf den Ethernet Ports im Ruhezustand auf allen Pis.

Agent	Schreiben KB/s	Lesen KB/s	Input/Outputit Ops/s
Dazzle	0,71 KB/s	0 KB/s	1,04 Ops/s
Tusk	0,18 KB/s	0 KB/s	0,02 Ops/s
Tinker	0,69 KB/s	0 KB/s	0,29 Ops/s
Lion	0,74 KB/s	0 KB/s	1,02 Ops/s
Agent ∅	0,58 KB/s	0 KB/s	0,59 Ops/s
Agent σ	0,23 KB/s	0 KB/s	0,45 Ops/s

Tabelle 4.3: I/O Zeiten im Ruhezustand auf den Pis.

Agents, neben den ausgehenden Informationen der Passive Checks, auch nach den in den Active Checks vereinbarten Informationen fragen. So es, wie die Tabelle 4.2 nahelegt, dass die durchschnittlichen Werte für den minimalen Traffic mit 2,55 Kilobit/s, durchschnittlichen mit 2,71 Kilobit/s und maximalen Traffic 3,41 Kilobit/s höher sind als für den eingehenden. Die Standardabweichung in den Tabellen zeigen die mögliche Abweichungen vom Durchschnitt an und sind ein Indikator dafür, ob einer der Hosts ein ungewöhnliches Arbeitsverhalten aufweist. In diesem Test sind diese jedoch sehr gering und es liegt somit nahe, dass alle Hosts einwandfrei funktionieren.

In der Tabelle 4.3 ist die Auslastung der Festplatte während kein Traffic auf den Hosts läuft. Es finden keine Leseoperationen statt und es werden durchschnittlich nur 0,58 KB/s geschrieben. In der Abb. 7.2 kann man auch klar erkennen das bis auf ein paar sehr kleine Schreiboperationen nichts passiert. Die Festplatte befindet sich primär in einem Idle Zustand.

In der Tabelle 4.4 ist die prozentuale Auslastung der CPU-Last zu sehen. Man sieht das die CPU hauptsächlich im Idle Zustand ist. Dies liegt hauptsächlich daran, dass es ausser den Grundfunktionen des Betriebssystems und der Zabbix Agents, keine Tasks zu bearbeiten gibt. Die aufgebrachte User Time entsteht durch den Zabbix Agent, welcher vom User Zabbix

Agent	Idle %	User Time %	System Time %	I/O wait Time %	Software IRQ %
Dazzle	96,70 %	1,07 %	1,84 %	0,03 %	0,10 %
Tusk	95,88 %	1,55 %	2,45 %	0,01 %	0,11 %
Tinker	96,96 %	1,07 %	1,86 %	0,02 %	0,10 %
Lion	96,99 %	1,07 %	1,84 %	0,03 %	0,07 %
Agent ∅	96,63 %	1,19 %	2,00 %	0,02 %	0,10 %
Agent σ	0,45 %	0,21 %	0,26 %	0,01 %	0,02 %

Tabelle 4.4: CPU Lastverteilung im Ruhezustand auf auf den Hosts.

durchgeführt werden. Dass die Werte der System und User Time so gering sind, ist ein gutes Zeichen. Daraus kann man nämlich schließen das nur die notwendigen Anwendungen am Laufen sind und die CPU für weitere Tests frei verfügbar ist.

4.2 20 Megabyte Testlauf

Der erste durchgeführte Test basiert auf 20 Megabyte Paketen. Dazu wurde das Hintergrundrauschen so eingestellt, dass die Größe der verschickten Pakete 20 Megabyte beträgt.

Wie man in Abb. 5.1 sehen kann, ist der durchschnittliche ausgehende Traffic auf dem Pi Dazzle 8,85 Mbit/s. Der eingehende Traffic beträgt durchschnittlich 8,13 Mbit/s. In der Tabelle 4.5 ist der Datenverkehr der einzelnen Hosts aufgelistet. Betrachtet wird der minimale, maximale und der durchschnittlich eingehende Traffic. Der maximal eingehende Datenstrom beträgt durchschnittlich 19,205 Mbit/s. Der minimal eingehende Traffic liegt bei durchschnittlich 28,45 Kbit/s. Die minimalen Werte entstehen, wenn der Pi am Ende oder am Anfang einer Paketübertragung ist. Die Agents erreichen jedoch nicht die minimalwerte, wie sie im Test ohne einen Datenaustausch erreicht werden, welcher der Traffic der durch die Kommunikation zwischen Agents und Server entsteht. Dies kann in den folgenden Tests in Abschnitte 4.3 und 4.4 auch so beobachtet werden. Die Standardabweichung der durchschnittlichen und der maximalen Last sind geringer als 1 Mbit/s. Aus dieser geringen Abweichung vom Durchschnitt kann man schließen, dass die Hosts gleichmäßig ausgelastet sind.

Da aber alle Hosts nicht nur Empfänger, sondern auch gleichzeitig Sender sind, wird auch der ausgehende Datenstrom betrachtet. Aus der Tabelle 4.6 kann man ablesen, dass durchschnittlich 8,56 Mbit/s von jedem einzelnen Host verschickt werden. Wie man sieht, beträgt die Standardabweichung σ vom ausgehenden Datenstrom 0,18 Mbit/s. Auch die Maximallast hat nur eine geringe Standardabweichung. Die Werte der Maximallast und des Durchschnitts ähneln sehr denen vom eingehenden Traffic. Der Wert der Minimallast des Hosts Lion weicht mit 3,03 Mbit/s jedoch deutlich ab, wodurch sich der Durchschnitt der Minimallast drastisch erhöht. Man kann auch beobachten, dass der Durchschnitt des maxmimal ausgehenden Traffics 4 Mbit/s geringer ist als der des eingehenden.

In der Abb. 7.5 sieht man, dass die Festplatte des Raspberry Pis konstant beschrieben wird.

Agent	Minimal Kb/s	Durchschnitt Mb/s	Maximal Mb/s
Dazzle	31,64 Kb/s	8,13 Mb/s	18,54 Mb/s
Tusk	29,11 Kb/s	8,51 Mb/s	19,15 Mb/s
Tinker	34,30 Kb/s	8,33 Mb/s	20,28 Mb/s
Lion	18,76 Kb/s	8,40 Mb/s	18,85 Mb/s
Agent ∅	28,45 Kb/s	8,34 Mb/s	19,20 Mb/s
Agents σ	5,89 Kb/s	0,14 Mb/s	0,66 Mb/s

Tabelle 4.5: Eingehender Traffic auf den Ethernet Ports bei 20 Megabyte Paketen auf allen Pis.

Agent	Minimal Kb/s	Durchschnitt Mb/s	Maximal Mb/s
Dazzle	628,48 Kb/s	8,85 Mb/s	15,02 Mb/s
Tusk	61,18 Kb/s	8,47 Mb/s	15,35 Mb/s
Tinker	421,76 Kb/s	8,52 Mb/s	13,95 Mb/s
Lion	3030,00 Kb/s	8,38 Mb/s	13,94 Mb/s
Agent ∅	1035,35 Kb/s	8,56 Mb/s	14,57 Mb/s
Agents σ	1169,37 Kb/s	0,18 Mb/s	0,63 Mb/s

Tabelle 4.6: Ausgehender Traffic auf den Ethernet Ports bei 20 MB Paketen auf allen Pis.

Agent	Schreiben KB/s	Lesen KB/s	Input/Output Ops/s
Dazzle	1,04 KB/s	0 KB/s	153,09 Ops/s
Tusk	1,11 KB/s	0 KB/s	42,40 Ops/s
Tinker	1,07 KB/s	0 KB/s	142,50 Ops/s
Lion	1,04 KB/s	0 KB/s	144,31 Ops/s
Agent ∅	1,07 KB/s	0 KB/s	120,73 Ops/s
Agent σ	0,03 KB/s	0 KB/s	45,31 Ops/s

Tabelle 4.7: I/O Zeiten bei Normalbetrieb auf den Pis.

Agent	Idle %	User Time %	System Time %	I/O wait Time %	Software IRQ %
Dazzle	25,45 %	37,71 %	22,32 %	1,42 %	16,10 %
Tusk	23,26 %	37,41 %	23,57 %	0,22 %	15,54 %
Tinker	23,78 %	35,99 %	22,80 %	1,65 %	15,77 %
Lion	22,82 %	35,99 %	23,15 %	1,73 %	15,84 %
Agent ∅	23,83 %	36,14 %	22,96 %	1,26 %	15,81 %
Agent σ	0,10 %	0,97 %	0,46 %	0,61 %	0,20 %

Tabelle 4.8: CPU Lastverteilung

Leseoperationen finden überhaupt nicht statt, das gilt auch für die anderen Agents wie man in der Tabelle 4.8 sieht. Im Durchschnitt werden 1,07 Kilobyte pro Sekunde geschrieben, was nicht annähernd der maximalen Schreibgeschwindigkeit der verwendeten SanDisk SD Karten entspricht, welche bei 30 Megabyte pro Sekunde liegt [Cor16]. Aus der Tabelle 4.7 lässt sich also schließen, dass die Festplatten der Agents kaum belastet werden.

Betrachtet man nun die Tabelle 4.8 spiegeln sich die Ergebnisse aus der Tabelle 4.7 wider. Die Zeit, die die CPU braucht um auf den Abschluss einer Lese-/Schreib Operation zu warten, hält sich sehr gering. So werden durchschnittlich 1,26 % der CPU Zeit für Lese-/Schreib Operationen verwendet. Der Großteil der CPU Zeit wird bei allen Hosts dafür verwendet, die Useranwendungen laufen zu lassen. Das entspricht im Schnitt 36,14 %. Die Idle Spalte zeigt an, wieviel Prozent der Prozessorleistung ohne eine Aufgabe war. Während die Systemzeit für die Verwaltung von Netzwerkaufgaben, wie dem Versenden und Empfangen von im Netzwerk verschickten Paketen, zuständig war. Wenn man nun die Werte der Standardabweichung der CPU betrachtet, bestätigt sich die aus der Tabelle 4.5 und der Tabelle 4.6 gewonnene Annahme, dass alle Hosts gleichmäßig belastet worden sind. Die Standardbweichung der Erwartungswerte übersteigt 1 % nicht, was ein Indikator dafür ist, dass die Prozessoren auf den Hosts in etwa dieselbe Last tragen.

Über eine Dauer von 12 Stunden wurden 10937 Pakete im Netzwerk verschickt. Die Menge der versendeten Daten in diesem Netzwerk beträgt 213,61 Gigabyte. Rechnet man dies auf eine Stunde herunter, erhält man 17,8 Gigabyte pro Stunde oder 911,41 Pakete pro Stunde. Wieder lässt sich eine ausgewogene Verteilung erkennen. Die Standardabweichung der erfolgreich verschickten Pakete liegt bei 15 Paketen und auch die Fehlerrate ist sehr gering. Ein Host hat sogar innerhalb von 12 Stunden kein einziges Paket erfolglos absenden können. Generell ist eine sehr geringe Menge an Daten verloren gegangen. Es haben Insgesamt 80 Megabyte ihr Ziel nicht ordnungsgemäß erreicht.

In der Tabelle 4.9 ist die Anzahl der verschickten und empfangenen Pakete aufgelistet. In den Zeilen stehen die sendenden Hosts, in den Spalten die empfangenden Hosts, sprich: Dazzle schickt an Tusk 685 Pakete während dem Gesamten Testlauf oder Tinker empfängt von Lion 679 Pakete. Die Tabellen in Abschnitt 4.3 und Abschnitt 4.4 lesen sich gleich.

Agent	Erfolgreich gesendet	Erfolglos gesendet	Erfolglos gesendet %	Verschickte GB
Dazzle	2731	1	0,04 %	53,34 GB
Tusk	2710	0	0,00 %	52,93 GB
Tinker	2751	2	0,07 %	53,73 GB
Lion	2745	1	0,04 %	53,61 GB
Summe	10923	4	0,03 %	213,61 GB
Agent ∅	2734,25	1	0,0375 %	53,40 GB
Agent σ	15,77	0,71	0,01 %	0,31 GB

Tabelle 4.9: Anzahl der gesendeten Pakete über einen Zeitraum von 12 Stunden.

Agent	Dazzle	Tusk	Tinker	Lion	Summe
Dazzle	659	685	678	707	2719
Tusk	668	701	678	661	2708
Tinker	692	659	659	724	2752
Lion	703	671	679	691	2744
Summe	2712	2716	2694	2694	10923

Tabelle 4.10: Die Anzahl der verschickten und der empfangenen 20 Megabyte Pakete pro Host.

4.3 200 Megabyte Testlauf

Der zweite durchgeführte Test basiert auf 200 Megabyte Paketen. Dazu wurde das Hintergrundrauschen so eingestellt, dass die Größe der verschickten Pakete 200 Megabyte beträgt.

Wie man in Abb. 7.7 sehen kann, ist der durchschnittlich ausgehende Traffic auf dem Pi Dazzle 8,49 Mbit/s. Der eingehende Traffic beträgt durchschnittlich 8,31 Mbit/s. In der Tabelle 4.11 ist der Datenverkehr der einzelnen Hosts aufgelistet. Betrachtet wird der minimale, maximale und der durchschnittlich eingehende Traffic. Der Maximal eingehende Datenstrom beträgt durchschnittlich 19,205 Mbit/s. Der minimal eingehende Traffic liegt bei durchschnittlich 28,45 Kbit/s. Die Standardabweichung der durchschnittlichen Last ist geringer als 1 Megabit/s, während die der Maximallast gestiegen ist. Diese liegt nun bei 1,07 Megabit/s. Daraus kann man schließen, dass einige Pis mehr belastet werden als andere. Denoch sind die Pis noch recht gleichmäßig ausgelastet wie die Auslastung der CPU in der Tabelle 4.14 nahelegt.

Aus der Tabelle 4.12 kann man ablesen, dass durchschnittlich 8,67 Mbit/s von jedem einzelnen Host verschickt werden. Wie man sieht, beträgt die Standardabweichung σ vom ausgehenden Datenstrom 0,34 Mbit/s. Auch die Maximallast hat nur eine geringe Standardabweichung. Die Werte der Maximallast und des Durchschnitts ähneln denen vom eingehenden Traffic. Diesmal ist jedoch die minimale Last vom Host Lion geringer als in Abschnitt 4.2, wodurch keine drastiche Standardabweichung auftritt. Auch der durchschnittlich ausgehende minimal Traffic hält sich sehr gering. Diesmal diesmal liegt er im Kilobit/s Bereich, statt wie in Abschnitt 4.2 im Megabyte Bereich.

In der Abb. 7.8 sieht man, dass die Festplatte des Hosts Dazzle nun nicht mehr konstant

Agent	Minimal Kb/s	Durchschnitt Mb/s	Maximal Mb/s
Dazzle	6,08 Kb/s	8,49 Mb/s	24,28 Mb/s
Tusk	5,82 Kb/s	8,31 Mb/s	25,23 Mb/s
Tinker	6,12 Kb/s	8,14 Mb/s	23,47 Mb/s
Lion	7,04 Kb/s	8,73 Mb/s	26,33 Mb/s
Agent ∅	6,05 Kb/s	8,41 Mb/s	24,74 Mb/s
Agents σ	0,14 Kb/s	0,22 Mb/s	1,07 Mb/s

Tabelle 4.11: Eingehender Traffic auf den Ethernet Ports bei 200 Megabyte Paketen auf allen Pis.

Agent	Minimal Kb/s	Durchschnitt Mb/s	Maximal Mb/s
Dazzle	7,62 Kb/s	8,46 Mb/s	21,11 Mb/s
Tusk	7,66 Kb/s	9,22 Mb/s	20,98 Mb/s
Tinker	7,34 Kb/s	8,67 Mb/s	20,73 Mb/s
Lion	7,63 Kb/s	8,32 Mb/s	23,05 Mb/s
Agent ∅	7,56 Kb/s	8,67 Mb/s	21,46 Mb/s
Agents σ	0,12 Kb/s	0,34 Mb/s	0,92 Mb/s

Tabelle 4.12: Ausgehender Traffic auf den Ethernet Ports bei 200 Megabyte Paketen auf allen Pis.

Agent	Schreiben KB/s	Lesen B/s	Input/Output Ops/s
Dazzle	2,71 KB/s	874,04 B/s	268,34 Ops/s
Tusk	2,54 KB/s	1160 B/s	122,64 Ops/s
Tinker	2,57 KB/s	825,75 B/s	234,48 Ops/s
Lion	2,76 KB/s	770,47 B/s	248,08 Ops/s
Agent ∅	2,65 KB/s	907,56 B/s	218,385 Ops/s
Agent σ	0,09 KB/s	150,27 B/s	56,576 Ops/s

Tabelle 4.13: I/O Zeiten bei 200 Megabyte auf den Pis

Agent	Idle %	User Time %	System Time %	I/O wait Time %	Software IRQ %
Dazzle	21,83 %	32,71 %	22,49 %	7,02 %	15,92 %
Tusk	25,57 %	34,01 %	23,57 %	1,20 %	15,63 %
Tinker	24,55 %	32,05 %	22,18 %	5,67 %	15,53 %
Lion	21,48 %	32,05 %	22,72 %	6,68 %	15,92 %
Agent ∅	23,35 %	33,66 %	22,74 %	5,14 %	15,75 %
Agent σ	1,75 %	0,98 %	0,52 %	2,33 %	0,17 %

Tabelle 4.14: CPU Lastverteilung bei 200 Megabyte Paketen im Netzwerk.

beschrieben wird. Stattdessen sieht man, dass inzwischen über längere Perioden ein Aussetzen des Schreibens auftritt. Im Durchschnitt werden 1.07 Kilobyte die Sekunde geschrieben, was nicht annähernd die maximale Schreibgeschwindigkeit der verwendeten SanDisk SD Karten ist. Diese liegt bei 30 Megabyte pro Sekunde [Cor16]. Aus der Tabelle 4.13 lässt sich also schließen, dass die Festplatten der Agents kaum belastet werden.

Betrachtet man nun die Tabelle 4.14, spiegeln sich die Ergebnisse aus der Tabelle 4.13 wider. Bei diesem Test kam es auch zu Leseoperationen und die Anzahl der Schreiboperationen haben sich verdoppelt. Das spiegelt sich auch in der Lese-oder Schreib Wartezeit wider. Die CPU braucht nun ungefähr das Fünffache der CPU Zeit um eine Lese, oder Schreib Operation abzuschliessen wie in Abschnitt 4.2 aufgezeigt. So wird durchschnittlich 5,14 % der CPU Leistung für Lese- und Schreiboperationen verwendet. Der Großteil der CPU Leistung wird bei allen Hosts dafür verwendet, die Useranwendungen laufen zu lassen. Im Schnitt nimmt das 33,66 % daraus folgt ungefähr 3 % weniger Leistung für die User Anwendungen als in Abschnitt 4.2. Die Zeit, die die CPU in einem Idle Zustand verbracht hat, ist ungefähr die selbe wie im Test in Abschnitt 4.2. Auch bei der System Time haben sich beide Werte im Schnitt kaum verändert. Wirft man jetzt einen Blick auf die Standardabweichungen der Hosts, sticht vor allem die I/O wait Time heraus. Der Host Tusk hat einen sehr geringen Wert in der I/O wait Time, während die anderen Hosts im Schnitt 5,14 % der CPU Leistung verbrauchen, liegt die von Tusk bei 1,20 %. Wie man weiter sehen kann, hat Tusk auch den höchsten Leerlaufprozess, System Time und User Time, auf der CPU.

Über eine Dauer von 12 Stunden wurden 1151 Pakete im Netzwerk verschickt. Die Menge der Daten, die insgesamt verschickt wurden, sind 224,80 Gigabyte in diesem Versuchsaufbau. Rechnet man dies auf eine Stunde herunter, kommt man auf 18,73 Gigabyte pro Stunde oder 95,91 Pakete pro Stunde. Wie man sieht ist die Menge an versendeten Daten beinahe gleich geblieben. Es wurde knapp 1 Gigabyte an Daten mehr verschickt, und dabei nur 10 % der Pakete verschickt wie in Abschnitt 4.2. Dazu kommt das überhaupt kein Paket verloren gegangen ist und alle Daten erfolgreich ihr Ziel erreicht haben.

Agent	Erfolgreich gesendet	Erfolglos gesendet	Erfolglos gesendet %	Verschickte GB
Dazzle	279	0	0 %	54,50 GB
Tusk	290	0	0 %	56,64 GB
Tinker	297	0	0 %	58,01 GB
Lion	285	0	0 %	55,66 GB
Summe	1151	0	0 %	224,80 GB
Agent ∅	287,75	0	0 %	56,20 GB
Agent σ	6,61	0	0 %	1,29 GB

Tabelle 4.15: Anzahl der gesendeten 200 Megabyte Pakete über einen Zeitraum von 12 Stunden.

Agent	Dazzle	Tusk	Tinker	Lion	Summe
Dazzle	75	77	78	69	299
Tusk	67	79	75	59	280
Tinker	70	73	56	77	276
Lion	68	75	70	83	296
Summe	280	304	279	288	1151

Tabelle 4.16: Die Anzahl der verschickten und der empfangenen 200 Megabyte Pakete pro Host.

4.4 2000 Megabyte Testlauf

Der dritte durchgeführte Test basiert auf 2000 Megabyte Paketen, dazu wurde das Hintergrundrauschen so eingestellt das die Größe der verschickten Pakete 2000 Megabyte beträgt.

Wie man in Abb. 7.10 sehen kann ist der durchschnittlich eingehende Traffic auf dem Pi Dazzle 6,39 Mbit/s. Der eingehende Traffic beträgt durchschnittlich 7,98 Mbit/s. Wie man sehen in den Tabellen 4.17 und 4.18 sehen kann ist jedoch der Traffic von Tusk sehr Abweichend vom Durchschnitt des restlichen Traffic. Der Host Tusk hat den höchsten durchschnittlich eingehenden Traffic und den höchsten Maximalwert Im Verlauf dieses Abschnittes wird noch weiter auf diesen Host eingegangen. Der minimal eingehende Traffic liegt bei durchschnittlich 28,45 Kbit/s. Die Standardabweichung der durchschnittlichen Last liegt bei 2,58 Megabit/s. Dies ist ein Vielfaches mehr als bei dem 20 Megabyte Testlauf und dem 200 Megabyte Testlauf. Die Maximallast hat sich zwischen den beiden vorhergehenden Tests eingependelt. Diese liegt bei 23,16 Megabit/s, im Gegensatz zu den vorhergangen Tests mit 19,20 Megabit/s und 24,83 Megabit/s. Der Durchschnitt der Minimalwerte liegt mit 6,15 leicht über dem der Minimalwerte aus der Tabelle 4.11.

Betrachtet man nun den ausgehenden Traffic der Hosts, kann man beobachten, dass der Host Tusk diesmal mit Abstand den geringsten ausgehenden Traffic hat. Der maximal ausgehende Traffic mit 19,84 Megabit/s sowie der durchschnittlich ausgehende Traffic mit 5,41 Megabit/s, liegen unter dem Durchschnitt von 23,09 Megabit/s und 8,17 Megabit/s. Auch die Standardabweichungen, welche in den vorherigen Versuchen sehr gering waren, sind bei diesem Test auf über 1 Megabit/s gestiegen. Obwohl die Standardabweichung der Minimalwerte

Agent	Minimal Kb/s	Durchschnitt Mb/s	Maximal Mb/s
Dazzle	6,14 Kb/s	6,39 Mb/s	23,54 Mb/s
Tusk	6,17 Kb/s	11,86 Mb/s	26,03 Mb/s
Tinker	6,13 Kb/s	5,02 Mb/s	21,24 Mb/s
Lion	6,14 Kb/s	8,63 Mb/s	24,98 Mb/s
Agent ∅	6,15 Kb/s	7,98 Mb/s	23,16 Mb/s
Agents σ	0,53 Kb/s	2,58 Mb/s	1,80 Mb/s

Tabelle 4.17: Eingehender Traffic auf den Ethernet Ports bei 2000 Megabyte Paketen auf allen Pis.

Agent	Minimal Kb/s	Durchschnitt Mb/s	Maximal Mb/s
Dazzle	7,71 Kb/s	8,48 Mb/s	22,33 Mb/s
Tusk	7,66 Kb/s	5,41 Mb/s	19,84 Mb/s
Tinker	7,65 Kb/s	9,57 Mb/s	25,22 Mb/s
Lion	7,63 Kb/s	9,20 Mb/s	24,97 Mb/s
Agent ∅	7,66 Kb/s	8,17 Mb/s	23,09 Mb/s
Agents σ	0,03 Kb/s	1,64 Mb/s	2,19 Mb/s

Tabelle 4.18: Ausgehender Traffic auf den Ethernet Ports bei 2000 Megabyte Paketen auf allen Pis.

Agent	Schreiben KB/s	Lesen KB/s	Input/Output Ops/s
Dazzle	2,21 KB/s	2,64 KB/s	257,88 Ops/s
Tusk	1,60 KB/s	2,86 KB/s	132,75 Ops/s
Tinker	1,89 KB/s	2,92 KB/s	257,25 Ops/s
Lion	2,76 KB/s	0,77 KB/s	248,08 Ops/s
Agent ∅	2,12 KB/s	2,30 KB/s	223,99 Ops/s
Agent σ	0,43 KB/s	0,89 KB/s	52,82 Ops/s

Tabelle 4.19: I/O Zeiten bei 2000 Megabyte auf den Pis.

in keinem anderen Test so niedrig war, sind bei diesem Testlauf die restlichen Standardabweichung die höchsten. Deshalb wird das vernachlässigt, da hier ein Netzwerk betrachtet wird, in dem viele Daten verschickt werden sollen. Ein Netzwerk im Ruhezustand, entspricht den Anforderungen. Deshalb kann man schlussfolgern, dass die Hosts nicht gleichmäßig belastet worden sind, wie die Tabellen 4.17 und 4.18 klar zeigen.

In der Abb. 7.11 sieht man, dass die Festplatte des Hosts Dazzle nun nicht mehr konstant beschrieben wird. Stattdessen ist ersichtlich, dass inzwischen über längere Perioden ein Aussetzen des Schreibens auftritt. Auch die Perioden in denen der Host liest sind gestiegen. Im Durchschnitt werden 1.07 Kilobyte/s geschrieben. Wieder bildet der Host Tusk das Schlusslicht mit 1,60 Kilobyte/s. Auch dieser Wert liegt unter dem Durchschnitt.

Die Tabelle Tabelle 4.20 zeigt, dass der Host Tusk einen sehr geringen Anteil der CPU, dem Leerlaufprozess gewidmet hat. Mit 12,05 % liegt dieser weit unter dem Durchschnitt. Das liegt unter anderem an der User Time, welche bei 42,43 % liegt und der System Time, die bei 26,74 % liegt. Beide Werte liegen deutlich über dem Durchschnitt der anderen Hosts, während die I/O wait Time deutlich unter dem Durchschnitt der anderen Hosts liegt. Die

Agent	Idle %	User Time %	System Time %	I/O wait Time %	Software IRQ %
Dazzle	29,48 %	29,07 %	21,12 %	7,33 %	12,88 %
Tusk	12,05 %	42,43 %	26,74 %	1,27 %	15,63 %
Tinker	29,16 %	28,97 %	21,29 %	8,11 %	12,41 %
Lion	29,16 %	30,68 %	22,08 %	6,68 %	16,55 %
Agent ∅	23,67 %	32,79 %	22,81 %	5,85 %	14,37 %
Agent σ	7,05 %	5,61 %	2,30 %	2,69 %	1,76 %

Tabelle 4.20: CPU Lastverteilung bei 2000 Megabyte Paketen im Netzwerk.

Agent	Erfolgreich gesendet	Erfolglos gesendet	Erfolglos gesendet %	Verschickte GB
Dazzle	22	3	12 %	42,97 GB
Tusk	14	4	22,22 %	27,34 GB
Tinker	22	3	12 %	42,97 GB
Lion	21	8	27,59 %	41,02 GB
Summe	79	18	22,78 %	154,30 GB
Agent ∅	19,75	4,50	18,45 %	38,57 GB
Agent σ	3,34	4,50	6,73 %	6,53 GB

Tabelle 4.21: Anzahl der gesendeten 2000 Megabyte Pakete über einen Zeitraum von 12 Stunden.

Gründe hierfür gehen eindeutig aus der Tabelle 4.22 hervor.

Über eine Dauer von 12 Stunden wurden 79 Pakete im Netzwerk verschickt. Die Menge der Daten, die insgesamt verschickt wurden, sind in diesem Versuchsaufbau 154,30 Gigabyte. Rechnet man dies auf eine Stunde herunter, kommt man auf 12,85 Gigabyte pro Stunde oder 6,59 Pakete pro Stunde. Wie man sieht, sind das knapp 80 Gigabyte weniger als in den vorhergehenden Tests. Dies liegt vorallem daran, dass an den Host Tusk keine Pakete erfolgreich versendet werden konnten. Jeder Versuch dem Host Tusk ein Paket zuzuschicken schlug fehl. Während alle anderen Hosts keine Pakete verloren haben. Der Grund für das Versagen des Verschickens der Pakete an den Host Tusk ist darauf zurückzuführen, dass der Host Tusk, laut Zabbix, nichtmehr genug freien Speicher zu Verfügung hatte. Da dies jedoch bei der Paketübertragung erst festgestellt wurde wenn der Vorgang fast abgeschlossen worden ist. Lief der Prozess weiter, bis der Host Tusk keine eingehenden Pakete mehr aktzeptierte und die bereits empfangenen Pakete wieder verwarf. Da der Prozess trotzdem eine Zeit lang durchgeführt worden ist und da der Sender erwartet hatte das die Datei aktzeptiert wird und den Sendeprozess erst spät beendete sind dann durch 80 Gigabyte verloren gegangen.

Agent	Dazzle	Tusk	Tinker	Lion	Summe
Dazzle	8	0	9	5	22
Tusk	10	0	5	7	22
Tinker	10	0	5	7	22
Lion	7	0	6	8	21
Summe	35	0	25	27	87

Tabelle 4.22: Die Anzahl der verschickten und der empfangenen Pakete pro Host.

5 Ergebnisse

In diesem Kapitel werden die Ergebnisse aus dem Kapitel 4 miteinander verglichen. Dabei wird als erstes die Anzahl der verschickten Pakete und die Menge der verschickten Bytes miteinander verglichen und die Zeiten die, ein Paket Paket zum erreichen des Zielsystems Benötigt. Es wird ein Blick auf den Ping im Netzwerk und die Auslastung auf den Ethernet Ports geworfen. Es wird viel Wert auf einen möglichst ausgeglichen Traffic an den Ports gelegt. Als zweites wird ein Blick auf die Auslastung des Prozessors gelegt. Vor allem wird versucht die Paketegröße zu finden, die die CPU am wenigsten in Anspruch nimmt, so das der Endnutzer einen möglichst Leistungsfähigen Host verwenden kann. Als drittes und letztes der im Kapitel 4 betrachteten Metriken kommt die Auslastung der Festplatte. Bevor es zur Schlussfolgerung kommt wird jedoch noch ein aufgetretener Fehler betrachtet der den Host Tusk in Abschnitt 4.4 betroffen hatte. Zum Schluss dieses Kapitels wird eine Schlussfolgerung aus den Ergebnissen gezogen.

Das Ziel in diesem Kapitel ist es, die Paketgröße zu ermitteln mit der man den höchsten Datenaustausch zwischen den Hosts erreichen kann, ohne das die Hosts Handlungsunfähig werden und Endnutzer noch auf den Hosts simple Arbeiten vollbringen können. Darunter wird gezählt das der Nutzer in der Lage ist den Rechner bedienen zu können und nicht das es auf dem Rechner möglich ist aufwändige Software zu verwenden, wie z.B: Photoshop oder ähnliches.

5.1 Versandte Daten

Da dieses Framework darauf ausgelegt ist den höchsten Datenverkehr zu bestimmen, ist die Auslastung der Ethernet Ports auf den jeweiligen Hosts der erste Indikator um zu messen, in welcher Paketgröße der höchste Austausch an Daten besteht. Dazu werden die durchschnittlichen Werte aus der Tabelle 5.1 miteinander verglichen. Diese Durchschnittswerte beziehen sich auf die Tabellen aus Kapitel 4, die den Traffic der Ports aufgelistet hatten. Bereits auf den ersten Blick ist erkennbar, dass sich der Traffic sich bei allen drei Tests nur um Kilobyte unterscheidet. Bei einer Paketgröße von 200 Megabyte ist jedoch ersichtlich, dass diese den höchsten eingehenden Traffic auf den Hosts erzeugt. Bei einer Paketgröße von 2000 Megabyte liegt dieser knapp unter 8 Megabit/s und ist damit die Paketgröße, die den geringsten eingehenden Traffic erzeugt. Bei den 20 Megabyte Tests kommen 8,34 Megabit/s durchschnittlich an. Dieser liegt im Mittelfeld im Vergleich zum eingehenden Traffic. Betrachtet man nun den

Tests	Eingehender Traffic Mb/s	Ausgehender Traffic Mb/s
∅ 20 Megabyte	8,34 Mb/s	8,55 Mb/s
∅ 200 Megabyte	8,42 Mb/s	8,67 Mb/s
∅ 2000 Megabyte	7,98 Mb/s	8,17 Mb/s

Tabelle 5.1: Durchschnittlicher eingehender und ausgehender Traffic der einzelnen Testfälle.

ausgehenden Traffic, so ist wieder der Test bei dem 200 Megabyte große Pakete verschickt worden sind an der Spitze mit 8,67 Megabit/s. Der 2000 Megabyte Test ist diesmal mit 8,17 Megabit/s über einen ausgehenden Traffic von 8 Megabit/s gekommen, bildet aber auch hier wieder das Schlusslicht der Übertragung. Bei einer Paketgröße von 20 Megabyte ist beträgt der ausgehende Traffic 8,55 Megabit/s und somit auch wieder zwischen den 200 Megabyte und den 2000 Megabyte Paketen.

Als zweiten Vergleichswert wird die Menge insgesamt verschickten Daten genommen. Da es nicht nur das Ziel ist, die Ethernet Ports der Hosts unter der größtmöglichen Last zu führen sondern, dass auch effektiv mehr Daten versendet werden sollen, mit denen dann ein anderes Hostsystem weiter arbeiten kann, ist auch die Menge der verschickten Daten ein entscheidendes Kriterium. Dazu wird die Menge an verschickten Daten im Netzwerk miteinander verglichen. Dazu wird in der Tabelle 5.2 die Anzahl der erfolgreich verschickten Pakete und verloren gegangenen Pakete, die Menge der verschickten Byte und der verloren gegangen Byte betrachtet. In der Tabelle 5.2 sind die Ergebnisse aus den Tabelle 4.9, Tabelle 4.15 und Tabelle 4.22 eingetragen. Bei diesem Vergleich ist es offensichtlich, welche der Paketgrößen die meisten Dateien verschicken kann. Hier sieht man, das die Größe der Pakete und Anzahl der verschickten Pakete Hand in Hand einhergehen. Auch sind vier verlorene Pakete eine sehr geringe Ausfallquote. Mit dieser geringen Ausfallquote von 0,04 % sind die 20 Megabyte Pakete die zweit zuverlßsigste Paketgröße in Bezug auf der Übertragungssicherheit. Jedoch ist am zuverlässigsten der Test der 200 Megabyte Dateien verschickt. In diesem Test ist kein einziges Paket verloren gegangen. Anders als im 2000 Megabyte Testfall. In diesem Test sind 18 Pakete verloren gegangen, was mit 17,14 % ein im Verhältnis mit den anderen beiden Tests sehr hoher Anteil an verlorenen Paketen ist. Insgesamt sind bei diesem Test 35,16 Gigabyte an Daten verloren gegangen. Das ist das 450-fache von dem, was im 20 Megabyte Test verloren gegangen ist. Damit ist der 2000 Megabyte Testfall nicht nur der unzuverläsigste von den drei Testfällen, was die Übertragung betrifft. Dieser Testfall schneidet auch im Vergleich der Menge an verschickten Daten am schlechtesten ab, so ist dieser mit 154,30 Gigabyte der Testfall der die wenigsten Informationen zwischen den Hosts austauschen konnte. Der Testfall mit den 20 Megabyte Paketen schneidet auch hier wieder am zweitbesten ab. Dieser hatte insgesamt 213,60 Gigabyte übertragen. Die meisten übertragenen Daten wurden vom 200 Megabyte Testfall erreicht, mit 224,80 Gigabyte.

Wie man aus der Tabelle 5.2 schließen kann wird im Testfall mit 20 Megabyte Paketen die höchste Frequenz an verschickten Paketen erreicht mit. Im Testfall mit 200 Megabyte Paketen ist Zeitspanne in der ein Paket verschickt wird am zweithöchsten, gefolgt vom 2000

Tests	Erfolgreich gesendet	Erfolglos gesendet	Verschickte GB	Verlorene Byte
20 Megabyte	10937	4	213,60 GB	80,00 MB
200 Megabyte	1151	0	224,80 GB	–
2000 Megabyte	87	18	154,30 GB	35,16 GB

Tabelle 5.2: Zusammenfassung der verschickten Pakete und Daten der einzelnen Testfälle.

Agent	∅20 Megabyte s	∅200 Megabyte s	∅2000 Megabyte s
Dazzle	15,84 s	157,65 s	1649,74 s
Tusk	15,95 s	147,40 s	1445,23 s
Tinker	15,65 s	155,40 s	1574,00 s
Lion	15,75 s	151,82 s	1467,96 s
Agent ∅	15,80 s	153,06 s	1534,23 s
Agent σ	0,11 s	3,88 s	82,52 s

Tabelle 5.3: Zeiterintervalle bis ein Paket erfolgreich sein Ziel erreicht hat.

Megabyte Testfall. In der Tabelle 5.3 kann wird genau dies aufgezeigt. Es wurden die Zeiten genommen, die in den Logfiles zwischen dem Versenden von zwei Paketen gespeichert wurde. Wie man man aus dieser Tabelle ablesen kann, wurde durchschnittlich alle 15,80 Sekunden ein 20 Megabyte Paket verschickt. Die Standardabweichung für 20 Megabyte Pakete liegt bei 0,11 Sekunden woraus man schließen kann, dass die Pakete zuverlässig in einem 15,80 $\pm$ 0,11 Sekunden Intervall versendet werden. Beim 200 Megabyte Testfall dauerte dies im Schnitt 151,82 Sekunden. Also zwei Minuten und 31,82 Sekunden was dem 9,61-fachem der Ubertragungsrate des 20 Megabyte Testfalls entspricht. Auch die Standardabweichung ist höher. Diese liegt bei 3,88 Sekunden, dass dem 35,27-fachen der Standardabweichung des 20 Megabyte Testfalls entspricht. Betrachtet man nun den 2000 Megabyte Testfall, bemerkt man, dass die Übertragungsdauer eines Paketes beim 10,02-fachen des 200 Megabyte Tests liegt oder verglichen mit dem 20 Megabyte Test das 97,10-fache der Übertragungszeit benötigt. Dies spiegelt sich auch so ungefähr in den verschickten Paketen wieder. Die Tabelle 5.2 zeigt auf, dass im 200 Megabyte Test ungefähr das die 10-fache Menge an Paketen verschickt wurde wie im 200 Megabyte Test. Um das jedoch so zu erreichen müssen auch die gescheiterten Paketübertragungen von dem 2000 Megabyte Test hinzugenommen werden. Dies liegt daran, dass der Host die Datenübertragung abgebrochen hat obwohl er schon einen Großteil der Daten empfangen hatte. Dasselbe Verhältnis herrscht auch zwischen dem 200 Megabyte Test und dem 20 Megabyte Test.

Als letztes wird der Ping der einzelnen Geräte in den verschiedenen Testfälle betrachtet. Obwohl der ICMP Ping, auf dem das Programm Pinger aufbaut nur eine Zuverlässige aussage gibt ob ein Host ansprechbar ist. Dies liegt daran das ICMP Ping eine andere Priorität hat als andere Pakete. So gibt es Aufschluss darüber ob die Antwort vom eingehende ICMP ECHO_REQUEST stark verzögert ist. Sollte die RTT ungewöhnlich hoch sein, kann es daran liegen, dass der gepingte Host eine sehr hohe CPU-Auslastung hat und den ICMP ECHO_REQUEST nicht beantworten kann. Wie man jedoch aus den Tabellen 5.4 bis 5.6

Agent	Dazzle	Tusk	Tinker	Lion	Durchschnitt
Dazzle	0,23 ms	0,81 ms	0,82 ms	0,99 ms	0,71 ms
Tusk	0,82 ms	0,23 ms	0,82 ms	0,96 ms	0,70 ms
Tinker	0,83 ms	0,81 ms	0,23 ms	0,99 ms	0,71 ms
Lion	0,97 ms	0,96 ms	0,97 ms	0,23 ms	0,78 ms
Durchschnitt	0,71 ms	0,70 ms	0,71 ms	0,79 ms	0,73 ms

Tabelle 5.4: Die RTT der Pis zueiander beim 20 Megabyte Testfall.

Agent	Dazzle	Tusk	Tinker	Lion	Durchschnitt
Dazzle	0,23 ms	0,85 ms	0,84 ms	1,00 ms	0,73 ms
Tusk	0,85 ms	0,24 ms	0,85 ms	1,02 ms	0,74 ms
Tinker	0,85 ms	0,83 ms	0,23 ms	1,00 ms	0,73 ms
Lion	0,99 ms	0,98 ms	0,99 ms	0,23 ms	0,80 ms
Durchschnitt	0,73 ms	0,73 ms	0,73 ms	0,81 ms	0,75 ms

Tabelle 5.5: Die RTT der Pis zueiander beim 200 Megabyte Testfall.

entnehmen kann, ist die Round Trip Time sehr gering. Beim 20 Megabyte Testfall beträgt der Durchschnitt der RTT 0,73 ms, während beim 200 Megabyte Testfall die RTT 0,75 ms beträgt, womit auch dieser Testfall die hochste RTT hat. Beim 2000 Megabyte Testfall liegt die RTT bei 0,73 ms und ist genauso niedrig wie beim 20 Megabyte Testfall. Auffällig ist jedoch das beim Host Lion die höchste RTT zu vermessen ist. Betrachtet man die Tabellen 4.8, 4.14 und 4.20 sieht man das Lion ausser, im 2000 Megabyte Test immer den geringsten Wert im CPU Leerlauf hat, dass die hohe RTT von Lion erklärt.

5.2 Prozessor Auslastung

Um die Annahme, die am Ende von Abschnitt 5.1 gemacht wurde, zu überprüfen, dass ein hohe RTT einer hohen Auslastung der CPU des Zielsystems entpricht, wird nun der Leerlauf den die CPUs haben, betrachtet. Da man über den Leerlauf Rückschlüsse über die Auslastung der CPU machen kann die Theorie die mit der RTT über den Host Lion aufgestellt wurde, dass dieser aufgrund der hohen RTT unter der höchsten Last stand. Wird mit den Werten aus der Tabelle Tabelle 5.7 bestägigt. Vergleicht man den Leerlauf von Lion mit denen der anderen Hosts, ist der Host Lion derjenige mit dem geringsten Leerlauf. Im 2000 Megabyte Testfall jedoch ist die CPU vom Host Tusk am meisten ausgelastet. Der Leerlauf beim Host

Agent	Dazzle	Tusk	Tinker	Lion	Durchschnitt
Dazzle	0,23 ms	0,81 ms	0,82 ms	0,99 ms	0,71 ms
Tusk	0,82 ms	0,23 ms	0,82 ms	0,99 ms	0,77 ms
Tinker	0,83 ms	0,80 ms	0,23 ms	0,99 ms	0,74 ms
Lion	0,97 ms	0,96 ms	0,97 ms	0,23 ms	0,78 ms
Durchschnitt	0,71 ms	0,70 ms	0,71 ms	0,80 ms	0,73 ms

Tabelle 5.6: Die RTT der Pis zueiander beim 2000 Megabyte Testfall.

Agent	20 Megabyte Idle	200 Megabyte Idle	2000 Megabyte Idle
Dazzle	25,45 %	21,83 %	29,48 %
Tusk	23,26 %	25,57 %	12,05 %
Tinker	23,78 %	24,55 %	29,16 %
Lion	22,82 %	21,48 %	29,16 %
Agent ∅	23,83 %	23,35 %	23,67 %
Agent σ	0,10 %	0,98 %	7,05 %

Tabelle 5.7: Leerlauf der CPUs im vergleich. Werte aus den Tabellen 4.8, 4.14 und 4.20.

Tusk liegt mit 12,05 % unter der Hälfte der anderen Hosts im Test. Dies ist auch der einzige Test in dem der Host Lion nicht die höchste Auslastung hat. Die hohe Auslastung des Hosts Tusk hängt aber auch mit dem Fehler zusammen, der während des Tests aufgetreten ist und in dem Abschnitt 4.4 und Abschnitt 5.4 schon aufgeführt ist.

Wie man auch direkt sehen kann sind die drei verbliebenen von den vier Hosts im 2000 Megabyte Test mit einem großen Abstand unausgelastet. Mit jeweils 29 % haben diese Hosts die geringste CPU-Last im Vergleich zu dem 20 Megabyte und 200 Megabyte Test. Jedoch ist Tusk mit 12,05 % der ausgelasteste Host in diesem Test. Dies wirkt sich auch auf die durchschnittliche Last im Test wieder. Mit 23,67 % liegt diesmal der 2000 Megabyte Test im Mittelfeld zwischen dem 20 Megabyte und 200 Megabyte Test. Bei 20 Megabyte Paketen sind Hosts im Test mit 23,83 % die mit dem höchsten Leerlaufprozess. Während beim 200 Megabyte Test mit 23,35 % der geringste Leerlaufprozess erreicht ist. Aus der Standardabweichung der einzelnen Tests kann man auch schlussfolgern, welcher der Tests eine möglichst gleichmäßige Belastung der Hosts hat. So werden beim 20 Megabyte Test, alle Hosts mit einer Standardabweichung von ± 0,10 % am gleichmäßigsten belastet. Gefolgt vom 200 Megabyte Test. Bei diesem Test ist auch noch eine geringe Standardabweichung gegeben mit ± 0,98 % sind. Auch in diesem Test sind die Hosts gleichmäßig belastet. Anders als beim 2000 Megabyte Test ist die CPU Lastenverteilung ist sehr ungleich. Mit ± 7,05 % ist die Lastverteilung am ungleichmäßigsten. Der Host Tusk hat in diesem Test die höchste Last von allen Host.

5.3 Festplatten Auslastung

In diesem Abschnitt wird die Auslastung der Festplatten betrachtet. Dabei wird die Metrik der Input Output Operations per Second hinzugezogen. Diese gibt an, wieviele Lese- oder Schreiboperationen in der Sekunde auf der Festplatte stattfinden. Je höher der Wert ist, desto mehr wird auf der Festplatte gelesen oder geschrieben. Daraus folgt jedoch auch, dass die CPU eine höhere I/O Wait Time hat und somit länger darauf warteti, dass die Festplatte mit einer Schreib- oder Leseoperation abgeschlossen hat. Deshalb wird als erstes die CPU I/O Wait Time betrachtet. Wie man aus der Tabelle 5.8 sehen kann ist beim 20 Megabyte Test die geringste Wartezeit mit 1,26 % für die CPU. Danach gefolgt von dem 200 Megabyte

Agent	20 Megabyte	200 Megabyte	2000 Megabyte
Dazzle	1,42 %	7,02 %	7,33 %
Tusk	0,22 %	1,20 %	1,27 %
Tinker	1,65 %	5,67 %	8,11 %
Lion	1,73 %	6,68 %	6,68 %
Agent ∅	1,26 %	5,14 %	5,85 %
Agent σ	0,20 %	2,33 %	2,69 %

Tabelle 5.8: CPU Wartezeit auf den Abschluss einer Lese- oder Schreiboperation, Werte aus den Tabellen 4.8, 4.14 und 4.20.

Agent	20 Megabyte	200 Megabyte	2000 Megabyte
Dazzle	153,09 Ops/s	263,34 Ops/s	257,88 Ops/s
Tusk	42,40 Ops/s	122,64 Ops/s	132,75 Ops/s
Tinker	142,50 Ops/s	234,48 Ops/s	257,25 Ops/s
Lion	144,31 Ops/s	248,08 Ops/s	248,08 Ops/s
Agent ∅	120,73 Ops/s	218,39 Ops/s	223,99 Ops/s
Agent σ	45,31 Ops/s	56,58 Ops/s	52,82 Ops/s

Tabelle 5.9: Lese- und Schreibzugriffe auf die Festplatten während der Testläufe, Werte aus den Tabellen 4.7, 4.13 und 4.19.

Testfall, dieser hat mit 5,03 % die zweithöchste I/O Wait Time. Beim 2000 Megabyte Test ist mit 5,85 % die CPU am längsten damit beschäftigt auf den Abschluss einer Lese- oder Schreiboperation zu warten. Jedoch ist die Differenz zwischen dem 200 und 2000 Megabyte Test nicht so signifikant, wie zwischen dem 20 Megabyte Test.

In der Tabelle 5.9 sind die jeweiligen Lese- und Schreiboperationen pro Sekunde der einzelnen Testfälle aufgelistet. Wie man sieht, passieren die geringsten I/O Operationen während dem 20 Megabyte Test. Das spiegelt sich auch in der I/O Wait Time der CPU wider. Mit durchschnittlich 120,73 Ops/s wird bei diesem Test am wenigstens gelesen oder geschrieben. Wie die Tabelle 4.7 auch zeigt, werden in diesem Testfall gar keine Daten gelesen. Das erklärt auch, wieso die Werte für die Lese- und Schreiboperationen so gering sind. Während beim 200 Megabyte Test die ersten Lese Operationen mit durchschnittlich 907,56 Byte/s stattfinden, wie man aus Tabelle 4.13 lesen kann. Dies erklärt auch den Anstieg der Lese- und Schreiboperationen der Durchschnitt der I/O Operationen/s zwischen den Tests mit 200 Megabyte Paketen und 2000 Megabyte Paketen liegen nahe beinander. Dasselbe ist auch beim Durchschnitt der CPU I/O Wait Time zu beobachten, jedoch haben sich die Verhältnisse verändert. Während beim 200 Megabyte Test mehr geschrieben als gelesen wurde ist dies beim Testfall mit 2000 Megabyte andersrum, in 2000 Megabyte Testfall wird mit durchschnittlich 2,30 KB/s gelesen nach Tabelle 4.19. Dafür ist die durchschnittliche Menge die geschrieben wird auf 2,12 KB/s, während die vom 200 Megabyte Test noch bei 2,65 KB/s lag.

5.4 Aufgetretene Fehler

Wie schon in Abschnitt 4.4 erwähnt kam es bei diesem Test zu einem Fehler bei einem der Hostsysteme. Alle Pakete, die zum Host verschickt werden sollten, haben ihr Ziel nicht erreicht. Dadurch sind sehr viele Pakete verloren gegangen. Insgesamt konnte vom Host Tusk kein einziges Paket angenommen werden. So sind alle fehlerhaften Pakete auf ihn zurückzuführen. Das Monitoring Tool Zabbix hat dazu selber eine Warnung ausgegeben. Diese besagt, dass der freie Festplattenspeicher zu mehr als 80 % belegt ist. Dies gibt einen ersten Hinweis darauf, dass die 2000 Megabyte Dateien an diesen Host nicht richtig gesendet werden können.

Abbildung 5.1: Fehlermeldung bezüglich der Festplatte von Tusk

Schaut man in die Logfiles und betrachtet die Zeit die zwischen dem Versenden von zwei Paketen vergeht sieht man, dass die Zeit die bis zum versenden des nächsten Packetes sehr hoch ist in diesem Fall 17 Minuten.

```
[Thu 16 Jun 19:36:11 CEST 2016]TRANSPORT FAILED
[Thu 16 Jun 19:53:53 CEST 2016]TRANSPORT SUCCESSFUL
```

Das Übertragen von erfolgreichen Paketen dauert nach der Tabelle 5.3 durchschnittlich 1534,23 Sekunden also ungefähr 25 Minuten. Mit diesem Wissen kann man die Schlussfolgerung ziehen, dass der Abbruch des verschickten Paketes an den Host Tusk erst sehr spät in der Übertragung passiert. SCP überprüft vor dem Versenden einer Datei nicht nach, ob genug Speicherplatz für diese Datei vorhanden ist. So kommt es auch, dass der Host Tusk auch im Test auf dem Ethernet Port eine eingehende Last aufweisen kann. Da SCP auf der SSH aufbaut welches das TCP Protokoll verwendet, werden die verschickten Dateien segmentiert. Für gewöhnlich sind solche Segmente 1500 Byte groß. Dies bedeutet, dass der Host Tusk solange 1500 Byte TCP-Segmente empfängt, bis bei diesem die Festplatte voll ist und keine weiteren Segmente mehr annehmen kann. Daraus folgt dann, dass die Datenübertragung abgebrochen wird, Zabbix hat für die verschickten Segmente Statistiken sammeln können, wie z.B: Den eingehenden Traffic und Belastung der CPU, obwohl der Host nie die komplette Datei empfangen hat.

5.5 Schlussfolgerung

Wie bei vielem in der Informatik ist es nicht möglich *das Beste* zu definieren und danach eine Auswahl zu treffen. Es muss ständig abgewogen werden was in einem Use Case am wichtigsten ist. Jeder der gezeigten Tests, hat den anderen Tests gegenüber Vor- und Nachteile. Deshalb betone ich an dieser Stelle nochmal die Zielsetzung die in Kapitel 1 definiert wurde.

> Das Ziel dieser Bachelorarbeit ist es herauszufinden, ob über die Paketgröße eine Optimierung der Performanz der Endgeräte und der Datenübertragung im Netzwerk möglich ist. Dabei wird versucht die größtmögliche Menge an Daten im Netzwerk auszutauschen ohne die Funktionalität eines der Hosts zu gefährden.

Wie man in Kapitel 4 sehen konnte, gibt es signifikante Unterschiede bei den in dieser Ausarbeitung durchgeführten Testfälle. Deshalb wird der Fokus erstmal darauf liegen, die Anforderung der Zielsetzung zu erfüllen. Mit dem Abschnitt der Zielsetzungi, das die Funktionalität der Hosts nicht gefährdet wird, fällt der 2000 Megabyte Testversuch schonmal als eine geeignete Paketgröße raus. Aufgrund der in Abschnitt 5.4 hohen Paketgröße kann der Host Tusk nicht fehlerfrei arbeiten. Seine CPU ist zu 88 % ausgelastet und Tusk kann keine empfangen. Dies alleine ist schon ein Kriterium, dass 2000 Megabyte Pakete ausschließt. Dass auch noch 35,16 Gigabyte an Daten im Netz verloren gegangen und insgesamt nur 154,30 Gigabyte verschickt wurden, erfüllt auch die zweite Anforderung nicht, nämlich möglichst viele Daten zu verschicken. Damit fällt die mögliche Auswahl auf die 20 Megabyte Dateien oder die 200 Megabyte Dateien. Dazu wird als erstes die Menge der verschickten Gigabyte miteianander verglichen. Aus der Tabelle 5.2 lässt sich ablesen, dass bei 200 Megabyte 224,80 Gigabyte an Daten verschickt worden sind und keine einziges Packet verloren gegangen ist. Beim 20 Megabyte Test wurden hingegen nur 213,60 Gigabyte verschickt. Dabei sind 80 Megabyte an Daten verloren gegangeni, was viel weniger als 0,01 % ist. Deshalb sind 200 Megabyte Testdateien fürs erste am geeigneitsten. Jedoch muss noch der zweite Punkt beachtet werden, nämlich dass die Hosts im System nicht komplett überlastet werden und noch simple Aufgaben erfüllen können. Dazu werden die Werte aus der Tabelle 5.7 miteinanander verglichen. Der Leerlaufprozess beim 20 Megabyte Test beträgt 23,83 % und beim 200 Megabyte Test % 23,35 %. Obwohl das heißt das, dass die Hosts im Test mit 20 Megabyte Dateien weniger belastet waren als im Test mit 200 Megabyte Dateien, ist der Unterschied sehr geringfügig. Deshalb hält man sich bei der Entscheidung, welche der Dateigrößen am besten geeignet ist daran, welche der beiden Testfälle die meisten Daten verschickt hat, welche der 200 Megabyte Test war.

Da aber, wie am Anfang dieses Abschnittes erwähnt, die optimale Dateigröße vom jeweiligen Use Case abhängig ist muss man von Use Case zu Use Case unterscheiden. Als Beispiel in dem die 20 Megabyte Datei von Vorteil wäre, ist in eine Netzwerkanwendung wo die Zeit ein wichtige Rolle spielt und es nicht auf Menge der Daten ankommt sondern das schnell Daten ankommen die 20 Megabyte Datei größe die bessere Wahl.

6 Fazit

Mit dieser Bachelorarbeit soll bewiesen werden, dass es möglich ist, für ein gegebenes Netzwerk eine optimale Paketgröße zu bestimmen. So dass im Netzwerk eine maximale Menge an Daten übertragen wird und die Auslastung der im Netzwerk angeschlossen Hosts nicht blockiert. Die gesammelten Daten aus den in Kapitel 2 vorgestellten Frameworks, den verschiedenen Tests aus Kapitel 4 und der Zusammenfassung dieser Daten in Kapitel 5 zeigen, dass es möglich ist, für einen definierten Verwendungszweck eines Netzwerkes eine optimale Paketgröße zu bestimmen.

Damit ist es also auch möglich Netzwerke zu testen und rauszufinden welche Paketgrößen am besten in ihnen geeignet sind. Da jedoch in dieser Ausarbeitung die Daten per Hand verarbeitet wurden. Anstatt die Tests mit einer vordefinierten Menge an Paketgrößen durchzuführen. Wäre eine sinnvolle Erweiterung des Frameworks, diesen kompletten Prozess zu automatisieren dabei sollte jedoch auch die Auswertung der so gesammelten Daten automatisiert werden.

Dazu muss jedoch vorher klar definiert sein welche Anforderungen an das Netzwerk gestellt werden. Soll das Netzwerk mit einer hohen Frequenz Pakete verschicken? Oder sollen die Endgeräte im Netzwerk keine hohe Belastung der CPU haben?

Desweiteren ist es mit den in dieser Bachelorarbeit geschaffenen Grundlagen auch möglich in die Richtung der Netzwerkfehlererkennung zu gehen. In Abschnitt 5.4 konnte man über die gesammelten Daten Rückschlusse auf einen Fehler machen, der die Arbeit im Netzwerk gestört hat und auch teilweise Einfluss auf die anderen Hosts genommen hatte. Mit weiteren Tests kann man – auch mit Hilfe des Frameworks – eine Fehleranalyse betreiben. Sammelt man genug Erfahrungswerte kann man so mit dem Framework Fehler in einem Netzwerk automatisch erkennen lassen und die Fehler direkt beheben.

7 Anhang

7.1 Test Ruhezustand

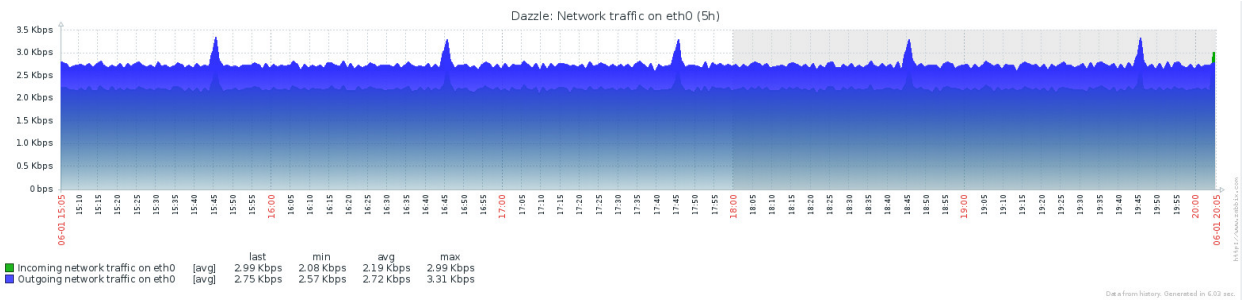

Abbildung 7.1: Traffic auf Eth0 ohne Hintergrundrauschen auf Dazzle.

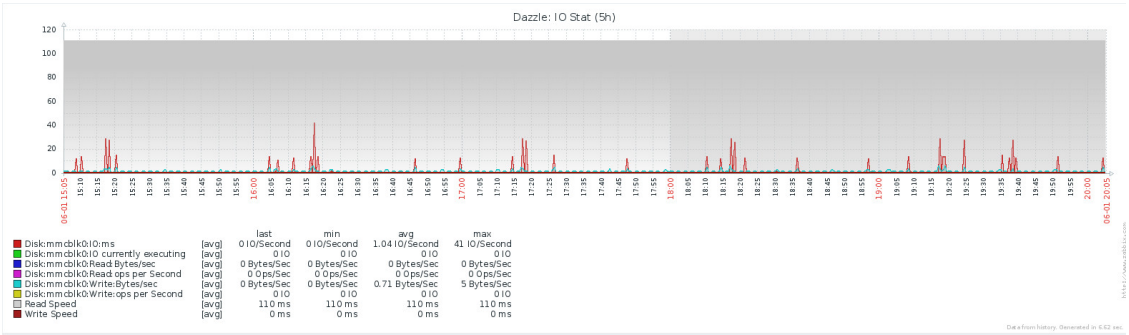

Abbildung 7.2: I/O Statistik von Dazzle ohne Hintergrundrauschen.

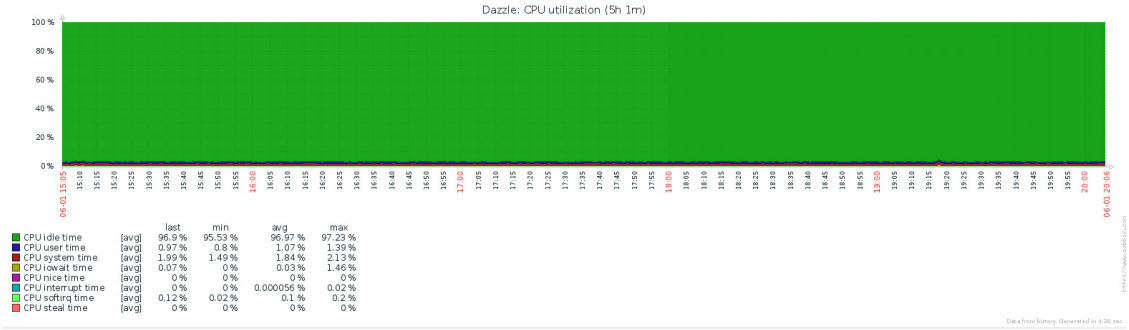

Abbildung 7.3: Prozentuale Lastverteilung auf der CPU von Dazzle, ohne Hintergrundrauschen.

7.2 Test 20 Megabyte Tests

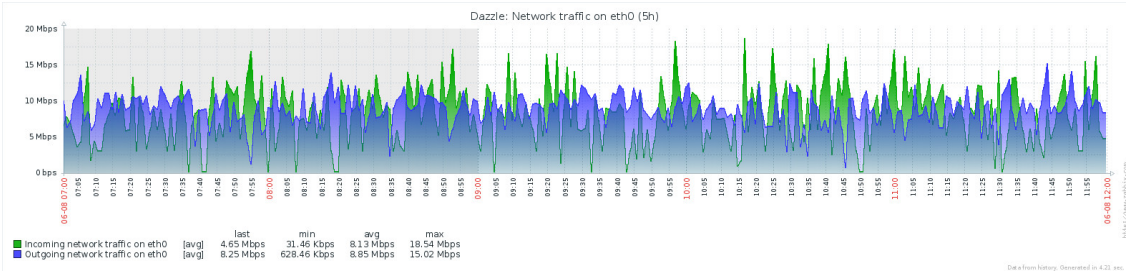

Abbildung 7.4: Traffic auf Eth0 bei 20 Megabyte Dateien auf Dazzle.

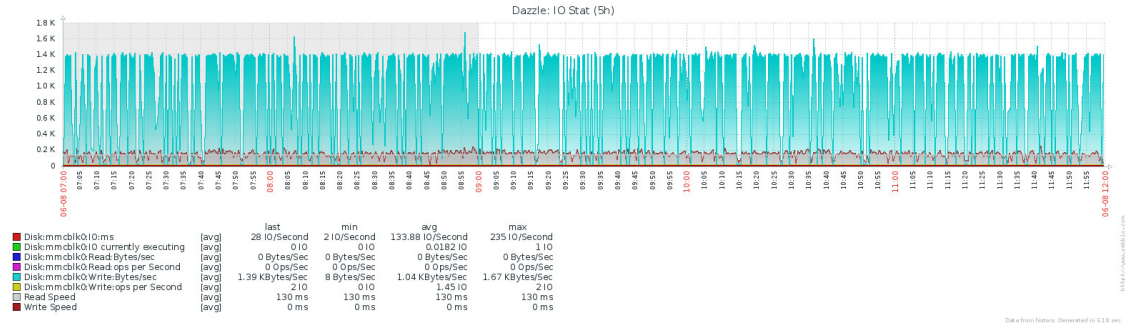

Abbildung 7.5: I/O Statistik von Dazzle beim 20 Megabyte Betrieb.

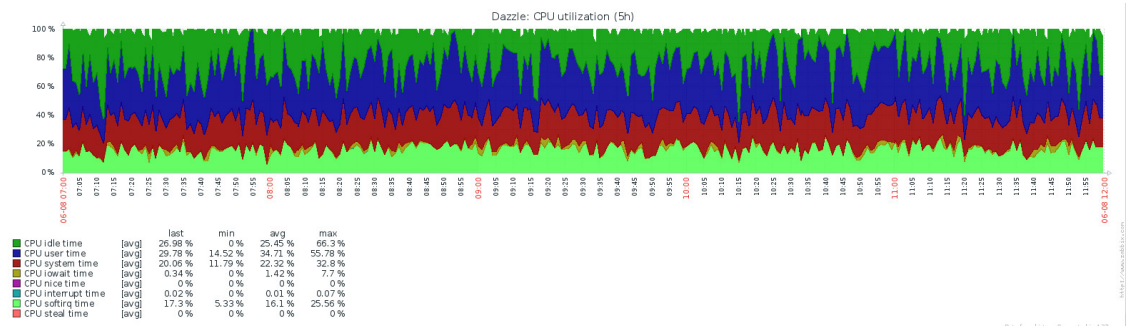

Abbildung 7.6: Prozentuale Lastverteilung auf der CPU bei 20 Megabyte Paketen von Dazzle.

7.3 200 Megabyte Tests

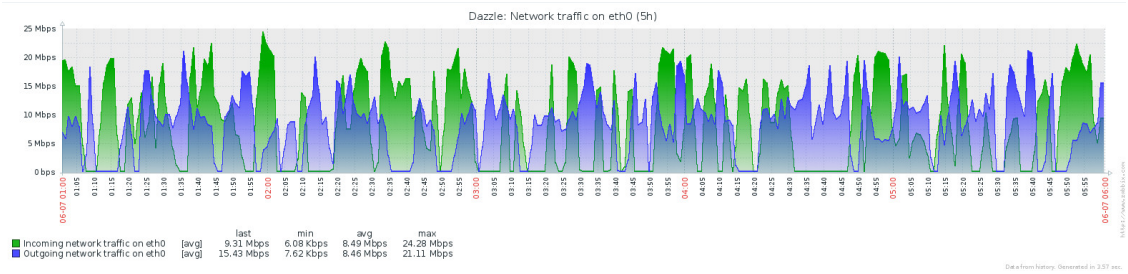

Abbildung 7.7: Traffic auf Eth0 bei 200 Megabyte Dateien auf Dazzle.

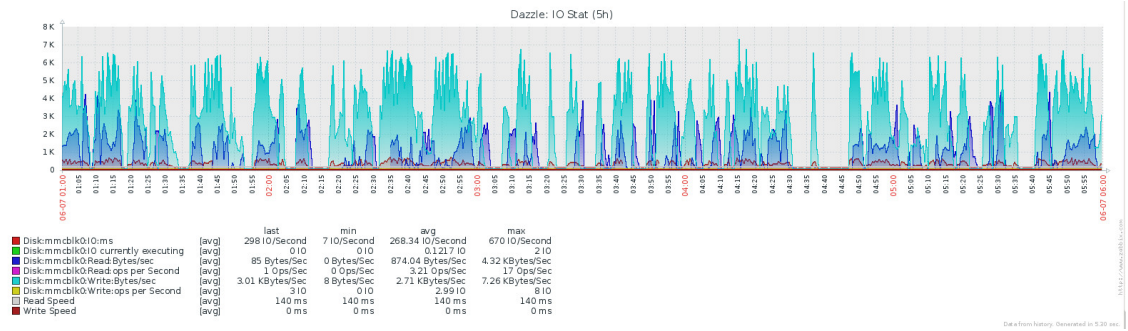

Abbildung 7.8: I/O Statistik von Dazzle beim 200 Megabyte Betrieb.

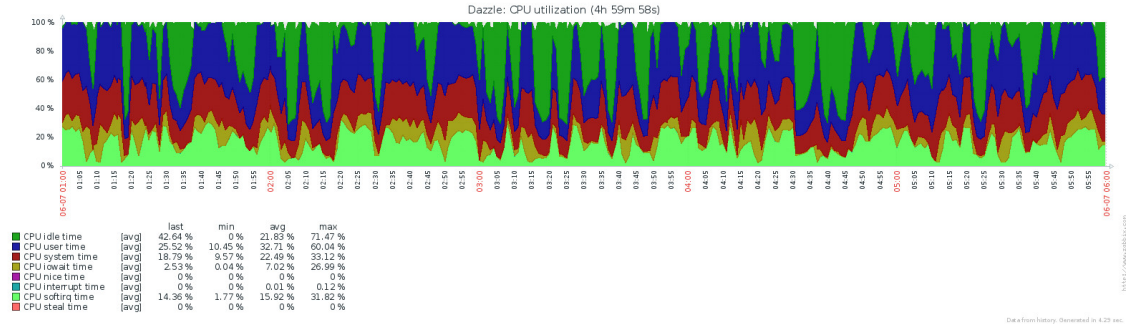

Abbildung 7.9: Prozentuale Lastverteilung auf der CPU bei 200 Megabyte Paketen von Dazzle.

7.4 2000 Megabyte Pakete

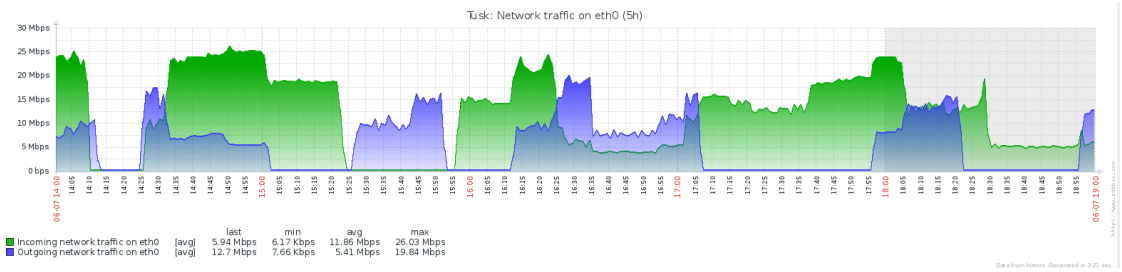

Abbildung 7.10: Traffic auf Eth0 bei 2000 Megabyte Dateien auf Dazzle.

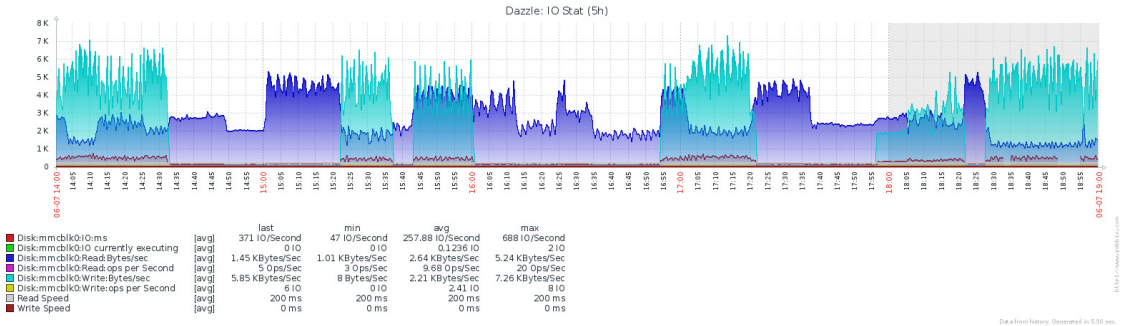

Abbildung 7.11: I/O Statistik von Dazzle beim 2000 Megabyte Betrieb.

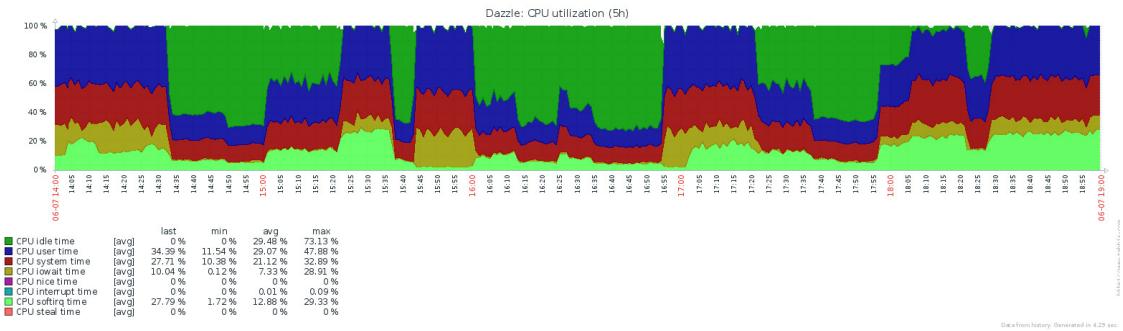

Abbildung 7.12: Prozentuale Lastverteilung auf der CPU bei 2000 Megabyte Paketen von Dazzle.

7.5 Inhalt der CD

Auf der CD sind folgende Inhalte:

- Die Logfiles zu den Tests,
- die selbsterstellten Programme,
- die Graphen der restlichen Hosts,
- eine Libre Office Calc Tabelle.

Literatur

[Ama16a] Amazon. *Amazon Raspberry Pi 1*. [Online; Stand 23. Juni 2015]. 2016. URL: https://www.amazon.de/Raspberry-Pi-RBCA000-Mainboard-1176JZF-S/dp/B008PT4GGC/ref=sr_1_1?ie=UTF8&qid=1466680050&sr=8-1&keywords=raspberry+pi+1.

[Ama16b] Amazon. *Amazon Raspberry Pi 2*. [Online; Stand 23. Juni 2015]. 2016. URL: https://www.amazon.de/Raspberry-Pi-quad-core-Cortex-A7-compatibility/dp/B00T2U7R7I/ref=sr_1_2?ie=UTF8&qid=1466680926&sr=8-2&keywords=raspberry+pi+1.

[Ama16c] Amazon. *Amazon Raspberry Pi 3*. [Online; Stand 23. Juni 2015]. 2016. URL: https://www.amazon.de/Raspberry-Pi-3-Model-B/dp/B01CEFWQFA/ref=sr_1_3?ie=UTF8&qid=1466680050&sr=8-3&keywords=raspberry+pi+1.

[Bor] Thorsten Bormer. „Secure Shell (ssh) Seminar: Simulationen mit User Mode Linux WS 2005/06“. In: ().

[Cal16] Berkeley University of California. *BOINC*. [Online; Stand 18. Juni 2016]. 2016. URL: http://boinc.berkeley.edu/.

[Cor16] SanDisk Corporation. *SD/SDHC/SDXC Specifications and Compatibility*. [Online; Stand 29. Mai 2016]. 2016. URL: http://kb.sandisk.com/app/answers/detail/a_id/2520/~/sd/sdhc/sdxc-specifications-and-compatibility.

[der15] derstandard.at. *Windows 10 f$\surd$ °r RaspberryPi2 : ErsteTestversionkostenloszumDownloo derstandard.at/2000015097563/Windows − 10 − fuer − Raspberry − Pi − 2 − Erste−Testversion−kostenlos−zum*. [Online; Stand 23. Juni 2016]. 2015. URL: http://derstandard.at/2000015097563/Windows-10-fuer-Raspberry-Pi-2-Erste-Testversion-kostenlos-zum.

[Kra16] Thorsten Kramm. *lab4.org*. [Online; Stand 11. Juni 2016]. 2016. URL: http://lab4.org/wiki/Zabbix_Items_Daten_per_Agent_sammeln#Welche_Daten_kann_der_Agent_liefern.3F.

[LLC16] Nagios Enterprises LLC. *Nagios*. [Online; Stand 29. Mai 2016]. 2016. URL: https://www.nagios.org/.

[Ors14] Lauren Orsini. *Raspberry Pi's Eben Upton: How We're Turning Everyone Into DIY Hackers*. [Online; Stand 22. Juni 2016]. 2014. URL: http://readwrite.com/2014/04/08/raspberry-pi-eben-upton-builders/.

[SIA] Zabbix SIA. *Zabbix*. [Online; Stand 11. Mai 2016]. URL: #https://www.zabbix.com/documentation/3.0/manual/concepts/agent#.

[SIA16a] Zabbix SIA. *Zabbix*. [Online; Stand 11. Mai 2016]. 2016. URL: http://www.zabbix.com/.

[SIA16b] Zabbix SIA. *Zabbix Anforderungen*. [Online; Stand . Juni 2016]. 2016. URL: https://www.zabbix.com/documentation/3.0/manual/installation/requirements.

[SIA16c] Zabbix SIA. *Zabbix Dokumentation*. [Online; Stand . Juni 2016]. 2016. URL: https://www.zabbix.com/documentation/3.0/manual/appendix/items/activepassive.

[SIA16d] Zabbix SIA. *Zabbix Share*. [Online; Stand . Juni 2016]. 2016. URL: https://share.zabbix.com/.

[Tan09] Andrew S. Tanenbaum. *Computernetzwerke*. 4. Überarb. Aufl., [Nachdr.] i - Informatik. München [u.a.]: Pearson Studium, 2009. ISBN: 9783827370464. URL: http://scans.hebis.de/HEBCGI/show.pl?21727132_toc.pdf.