h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt

- Fachbereich Informatik -

Name der Arbeit

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

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

> Dies ist ein Zitat.

verstand, scheinen nun doch vorueber zu Dies ist der Text sein.
siehe: http://janeden.net/die-praeambel

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

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. Inzwischen haben alle die Möglichkeit auf das Internet, welches über den gesamten Globus verteilt ist, zuzugreifen. Jedoch ist auch das Internet nur ein Netzwerk aus Netzwerken. 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. Dadurch 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 der Bestimmung einer optimalen Paketgröße

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.

1.3 Zielsetzung dieser Arbeit

Das Ziel dieser Bachelorarbeit ist es herauszufinden, ob über die Paketgröße eine Optimierung der Performanz der Endgeräte und der Datenübertragung im Netzwerk möglich ist.

1.4 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, 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 **??** 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.

Die beiden verwendeten Switches Arbeiten mit 100 oder 1000 Mbit/s, doch aufgrund der verwendeten Ethernet Ports auf den Raspberry Pis, welche eine Ubertrangsungrate 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 Aufbau der Software

Das Testframework besteht aus drei verschiedenen Teilen.

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

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

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

2.2.1 Zabbix

Zabbix ist ein Open Source Netzwerk Monitoring System. Die erste Version wurde von Alexei Vladishev entwickelt [SIA16a]. Ein weiterer bekannter Vertreter der Netzwerk Monitor Systeme 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. Welche 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 manuell die Templates 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, die Informationen an den Server weiterleiten, die vom Server gefordert werden. In den späteren Kapiteln wird der Hauptfokus auf der Auslastung der Festplatte, CPU und des Ethernet Ports liegen. Einen genaueren Einblick in den Zabbix Agent gibt es in Abschnitt 3.2.

2.2.2 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 [Bor] auf. So wird eine TCP Verbindung zum angesprochenen Host aufgebaut wird 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: Es 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 es handelsübliche Rechner haben. 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 Ping Befehl 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 Ping befehl ausgeführt. Über die Pipe wird dieser jedoch weitergeleitet und in einer Schleife weiterverarbeitet. Neben der Meldung, die vom Befehl Ping Befehl kommt, wird noch ein Datum vorgestellt. Diese ganze Meldung wird dann in einer Logfile Datei abgespeichert.

2.3 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 Boot Vorgang 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 Konfigu-

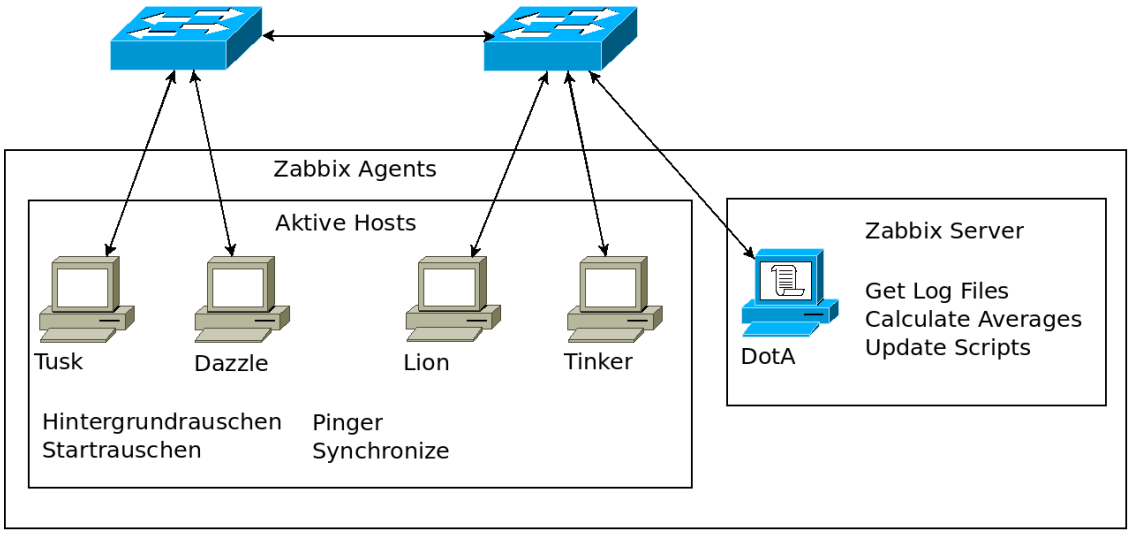

Abbildung 2.1: Aufbau des Netzwerks

rationsdatei von den Raspberry Pis ausgeführt. Deshalb gibt es auch keine Probleme die Uhrzeit zu setzen, da dieses Programm als Super User ausgeführt wird. Nach dem dieses Programm erfolgreich ausgeführt wurde, startet das Startrauschen Skript. Dieses Programm ermöglicht einen zeitversetzten von Hintergrundrauschen und Pinger. In Hintergrundrauschen müssen jedoch immer kleine Veränderungen vorgenommen werden. So muss je nach durchzuführendem Test eine Zeile umgeschrieben werden.

```
SIZE=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 einen Blockgröße und eine Anzahl an zu schreibenden Blöcken 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 dem 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. 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:

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 Ergebniss Zabbix3.0.0rc.
5. Die TCP Verbindung wird geschlossen.

[SIA16c]. Passive Checks und Active Checks unterscheiden sich darin, wer den Request auslöst. Bei einem Passiv Check sind die Hosts selber 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.

- 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).

[SIA16b]. 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 Perfomance* diese betrachtet zwei Dinge, einmal die sogenannte Queue, welche angibt wie viele von den Agents übermittelten Werten darauf warten vom Zabbix Server verarbeitet zu werden und die verarbeiteten Werte pro Sekunde. Diese Werte 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 auch 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 Sicherheits risiko 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:

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

[SIA]. 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, eingestellt werden. 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 hatte die das überwachen von der Festplattenauslastung gewährleistet. 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 den zugehörigen Werten $1 und 4 sind Variablen, die für die Verarbeitung notwendig sind. Der in diesem Beispiel genannte User Parameter liest die Lese-Operationen 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 4.1, **??** und **??** werden in **??** miteinander verglichen.

4.1 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. 7.1 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 4.1 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 keine Pakete empfängt und nur noch mit dem Zabbix Server kommuniziert. Dies kann in den folgenden Tests 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 4.2 kann man ablesen, dass durchschnittlich 8,56 Mbit/s von jedem einzelnen Host verschickt werden. Wie man sieht, beträgt

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	34,3 Kb/s	8,33 Mb/s	20,28 Mb/s
Lion	18,76 Kb/s	8,4 Mb/s	18,85 Mb/s
Agent ∅	28,4525 Kb/s	8,3425 Mb/s	19,205 Mb/s
Agents σ	5,88919084 Kb/s	0,138451255 Mb/s	0,6570578361 Mb/s

Tabelle 4.1: 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 Kb/s	8,38 Mb/s	13,94 Mb/s
Agent ∅	1035,35 Kb/s	8,5555 Mb/s	14,565 Mb/s
Agents σ	1169,3664626947 Kb/s	0,177552809 Mb/s	0,6308922253 Mb/s

Tabelle 4.2: 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	42,4 Ops/s
Tinker	1,07 KB/s	0 KB/s	142,5 Ops/s
Lion	1,04 KB/s	0 KB/s	144,31 Ops/s
Agent ∅	1,065 KB/s	0 KB/s	120,725 Ops/s
Agent σ	0,028722813 KB/s	0 KB/s	45,3117 Ops/s

Tabelle 4.3: I/O Zeiten bei Normalbetrieb auf den Pis

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 der 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 beobachtenn, dass der Durchschnitt des maxmimal ausgehenden Traffics 4 Mbit/s geringer ist als der des eingehenden.

In der Abb. 7.2 sieht man, dass die Festplatte des Raspberry Pis konstant beschrieben wird. Leseoperationen finden überhaupt nicht statt. Im Durchschnitt werden 1,07 Kilobytes pro Sekunde geschrieben, was nicht annährend der Maximalen Schreibgeschwindigkeit der verwendeten SanDisk SD Karten entspricht, welche bei 30 Megabyte pro Sekunde liegt [Cor16]. Aus der Tabelle 4.3 lässt sich also schließen, dass die Festplatten der Agents kaum belastet werden.

Betrachtet man nun die Tabelle 4.4 spiegeln sich die Ergebnisse aus der Tabelle 4.3 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

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 4.4: CPU Last Verteilung

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	1093	4	0,03 %	213,61 GB
Agent ∅	2734,25	1	0,0375 %	53,40 GB
Agent σ	15,76983	0,70711	0,01427 %	0,308004 GB

Tabelle 4.5: Anzahl der gesendeten Pakete über einen Zeitraum von 12 Stunden.

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 derTabelle 4.1 und der Tabelle 4.2 gewonnene Annahme, dass nämlich das 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.

Es wurde über eine Dauer von 12 Stunden 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.

4.2 200 Megabyte Testlauf

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

Wie man in Abb. 7.1 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 Ta-

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 4.6: Die Anzahl der verschickten und der empfangen Pakete pro Host.

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 4.7: 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 4.8: Ausgehender Traffic auf den Ethernet Ports bei 200 Megabyte Paketen auf allen Pis.

belle 4.7 ist der Datenverkehr der einzelnen Hosts aufgelistet, betrachtet wird der minimale, maximale und der durchschnittlich eingehende Traffic. Der Maximal eingehende Datenstrom ist 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 das einige Pis, mehr belastet werden als andere. Was aber noch keine drastischen Werte sind darauf hinweisen das bei der Verteilung der Netzwerk Last ein Pi, einen zu höheren Aufwand hat die eingehenden Daten zu verarbeiten.

Aus der Tabelle 4.8 kann man ablesen das 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 der vom eingehenden Traffic. Diesmal ist die jedoch die minimale Last vom Host Lion geringer als in **??**. Wodurch kein drastiche Standardabweichung Eintritt und auch der Durchschnittlich ausgehende minimal Traffic sich sehr gering hält, diesmal im Kilobit/s Bereich, statt wie in **??** im Megabyte Bereich.

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 4.9: 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 4.10: CPU Last Verteilung bei 200 Megabyte Paketen im Netzwerk

In der Abb. 7.2 sieht man das die Festplatte des Hosts Dazzle nun nicht mehr konstant beschrieben wird stattdessen sieht man das inzwischen über länger Perioden ein aussetzen des Schreibens auftritt. durchschnitt werden 1.07 Kilobytes die Sekunde geschrieben. Was nicht annährend die Maximale Schreibgeschwindigkeit der verwendeten SanDisk SD Karten ist, welche bei 30 Megabyte pro Sekunde [Cor16] liegt. Aus der Tabelle 4.9 lässt sich also schließen das die Festplatten der Agents kaum belastet werden.

Betrachtet man nun die Tabelle 4.10 spiegeln sich die Ergebnisse aus der Tabelle 4.9 wieder. Bei diesem Test kam es auch zu Leseoperationen und die Anzahl der Schreiboperationen haben sich verdoppelt. Das spiegelt sich auch in der I/O wait Time wieder. Die CPU braucht nun ungefähr das fünffache der, CPU-Zeit um um eine Lese, oder Schreib Operation abzuschliessen, als in **??**. So wird durchschnittlich 5,14 % der CPU Leistung für Lese und Schreib Operationen verwendet. Der Großteil der CPU Leistung wird bei allen Hosts, dafür verwendet die User Anwendungen laufen zu lassen, im Schnitt 33,66 %, welches ungefähr 3 % weniger Leistung für die User Anwendungen ist als in **??**. Die Zeit die die CPU in einem Idle Zustand verbracht hat ist ungefähr gleich zu den dem Test in **??** genauso wie die System Time, beide Werte haben sich im Durchschnitt kaum verändert. Wirft man jetzt einen Blick auf die Standardabweichungen der Hosts, sieht sticht vorallem die I/O Wait time heraus. Der Host Tusk, hat einen sehr geringen Wert in der I/O Wait time. Währen die anderen Hosts im Schnitt 5,14 % der CPU Leistung liegt die von Tusk bei 1,20 %. Wie man auch sehen kann hat Tusk auch den höchste Leerlauf auf der CPU, System Time und User Time.

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 runter kommt man auf 18,73 Gigabyte pro Stunde oder

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
		gesendet		

Tabelle 4.11: Anzahl der gesendeten 200 Megabyte Pakete über einen Zeitraum von 12 Stunden

95,91 Pakete pro Stunde. Wie man sieht ist die Menge an versendeten Daten Daten beinahe gleich geblieben. Es wurde knapp 1 Gigabyte an Daten mehr verschickt, und dabei nur 10 % der Pakete verschickt wie in **??**. Dazu kommt das überhaupt kein Paket verloren gegangen ist und alle Daten erfolgreich ihr Ziel erreicht haben.

4.3 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. 7.1 sehen kann ist der durchschnittlich eingehende Traffic auf dem Pi Dazzle 6,39 Mbit/s. Der eingehende Traffic beträgt durchschnittlich 7,98 Mbit/s. In der Tabelle 4.12 Wie man sehen kann ist jedoch der Traffic von Tusk sehr Abweichend vom Durchschnitt des restlichen Traffic. Der Host Tusk, hat den höchsten durchschnittlichen eingehenden Traffic und den höchsten Maximal wert 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 verhorgehenden Tests eingependelt, diese liegt bei 23,16 Megabit/s, anders als in 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 Tabelle 4.7.

Betrachtet man nun den ausgehenden Traffic der Hosts, kann man beobachten das der Host Tusk diesmal mit Abstand den geringsten Ausgehenden Trafic hat. Der Maximal ausgehende 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 Minimal Werte in keinem anderen Test so niedrig waren, sind bei diesem Testlauf. Dies ist jedoch zu vernachlässigen da hier ein Netzwerk betrachtet wird in dem viele Daten verschickt werden

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 4.12: 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 4.13: 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 4.14: I/O Zeiten bei 2000 Megabyte auf den Pis

sollen und nicht eines welches sich. in einem Leerlaufzustand befindet. Deshalb kann man Schlussfolgern das die Hosts nicht gleichmäßig belastet worden sind. Wie Tabelle 4.12 und **??** klar zeigen.

In der Abb. 7.2 sieht man das die Festplatte des Hosts Dazzle nun nicht mehr konstant beschrieben wird stattdessen sieht man das inzwischen über länger 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.

In der Tabelle 4.15 sieht man das der Host Tusk einen sehr geringen Idle Anteil der CPU hat. Mit 12,05 % liegt dieser weit unter dem Durchschnitt, dies liegt unter anderem an der User Time, diese liegt bei 42,43 %. und der System Time die bei 26,74 % liegt. Beide Werte liegen deutlich über dem Durchschnitt der anderen Hosts. Währen die I/O Wait Time deutlich unter dem Durchschnitt der anderen Hosts, liegt. Woran das liegt sieht man in der Tabelle 4.17.

Über eine dauer von 12 Stunden wurden 79 Pakete im Netzwerk verschickt. Die Menge

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 4.15: CPU Last Verteilung 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
		gesendet		

Tabelle 4.16: Anzahl der gesendeten 2000 Megabyte Pakete über einen Zeitraum von 12 Stunden

der Daten die insgesamt verschickt wurden sind 154,30 Gigabyte in diesem Versuchsaufbau. Rechnet man dies auf eine Stunde runter 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. Das 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 verschicken der Pakete an den Host Tusk ist darauf zurückzu"fuhren, das der Host Tusk, laut Zabbix nichtmehr genug freien Speicher zu verfügung hatte da bei der Paketübertragung das erst festgestellt wurde wenn der Vorgang fast abgeschlossen worden ist. Konnte der sendete Host, den Vorgang nicht früh genug abbrechen und sich einen neuen Host aussuchen, dadurch sind dann rund 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 4.17: Die Anzahl der verschickten und der empfangen Pakete pro Host.

5 Ergebnisse

5.1 Versandte Daten

5.2 Aufgetretene Fehler

5.3 Schlussfolgerung

6 Fazit

7 Anhang

7.1 20 Megabyte Test

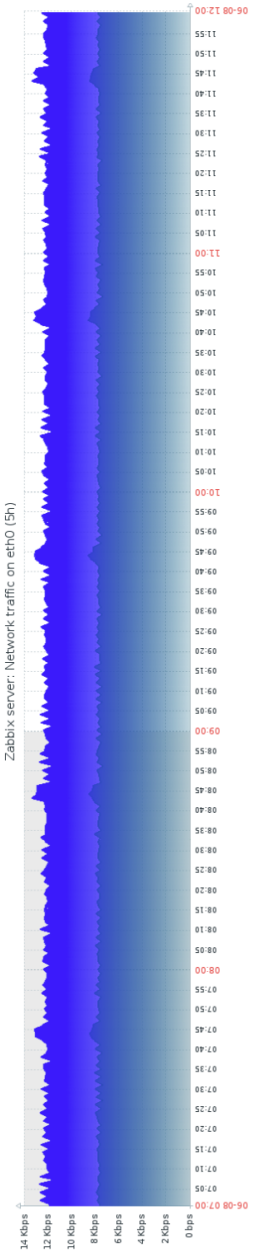

Abbildung 7.1: Traffic auf Eth0 bei 20 Megabyte auf Dazzle

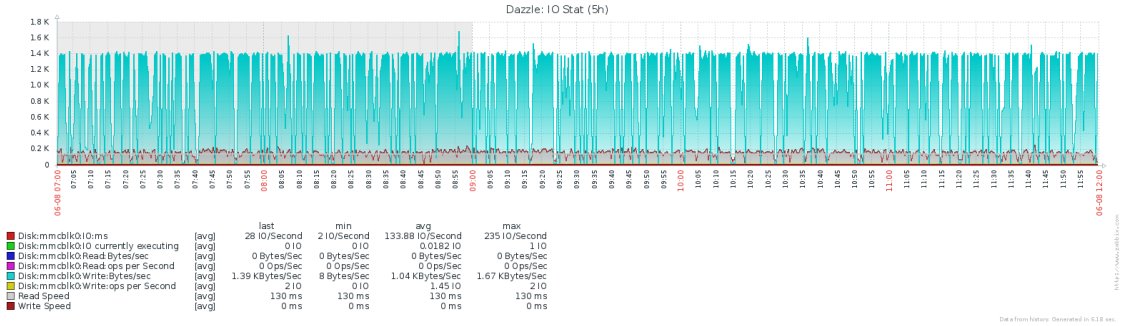

Abbildung 7.2: I/O Statistik von Dazzle beim 20 Megabyte Betrieb

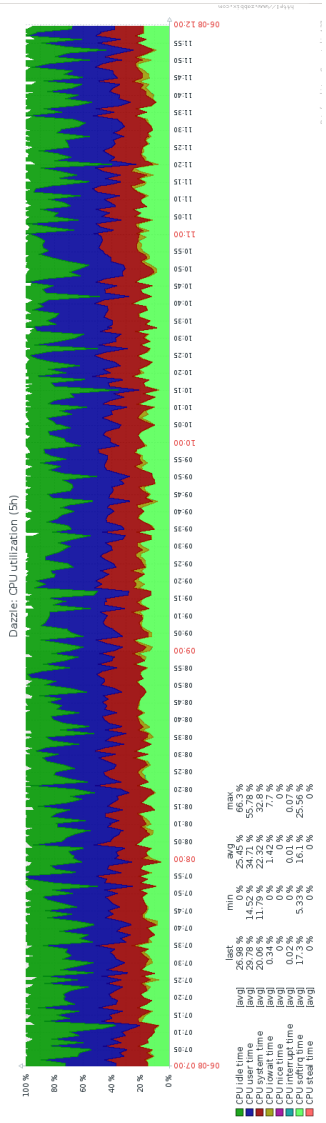

Abbildung 7.3: Prozentuale Lastverteilung auf der CPU von Dazzle

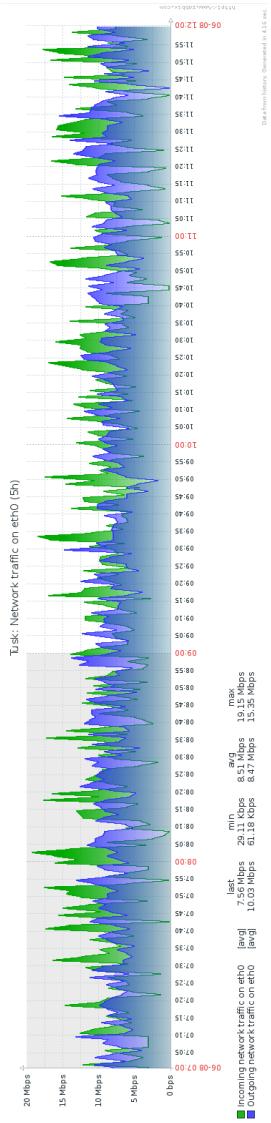

Abbildung 7.4: Traffic auf Eth0 bei 20 Megabyte auf Tusk

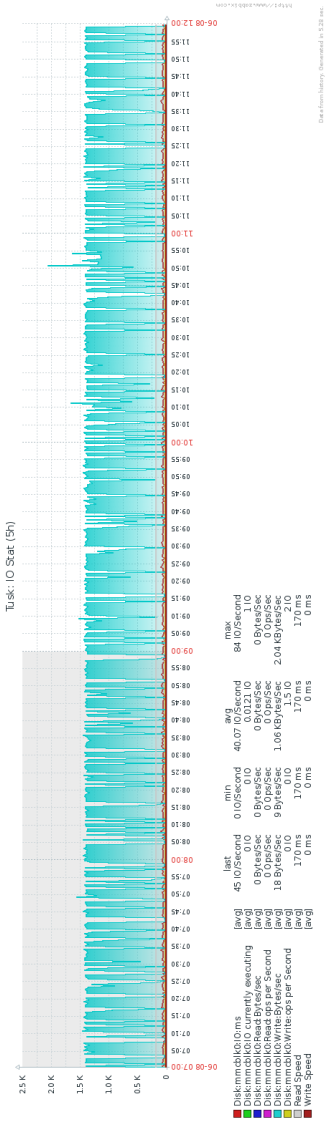

Abbildung 7.5: I/O Stats von Tusk beim 20 Megabyte Netzwerk betrieb

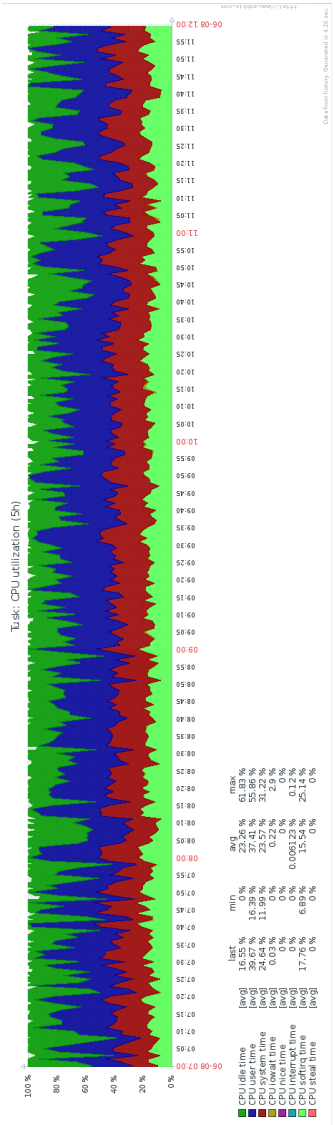

Abbildung 7.6: Prozentuale Lastverteilung auf der CPU von Tusk

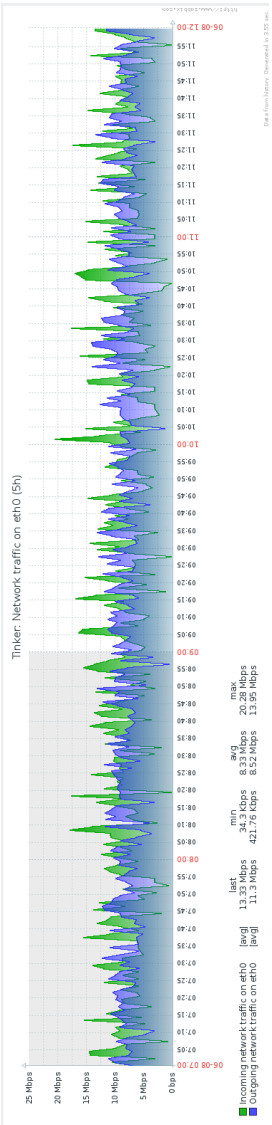

Abbildung 7.7: Traffic auf Eth0 bei 20 Megabyte auf Tinker

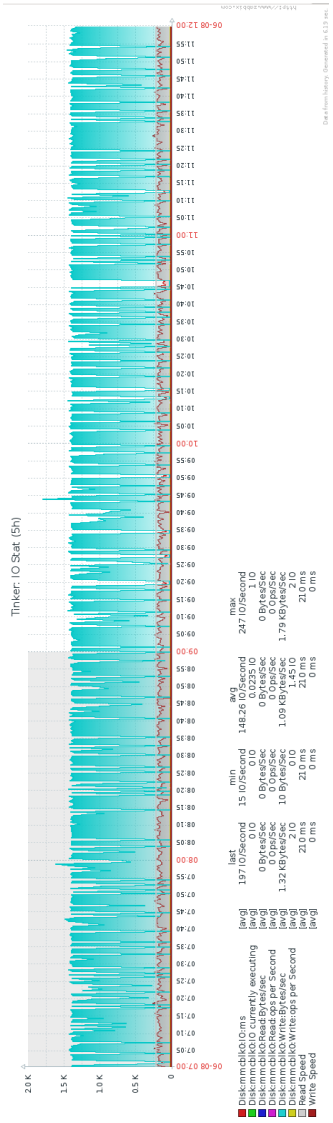

Abbildung 7.8: I/O Stats von Tinker beim 20 Megabyte Netzwerk betrieb

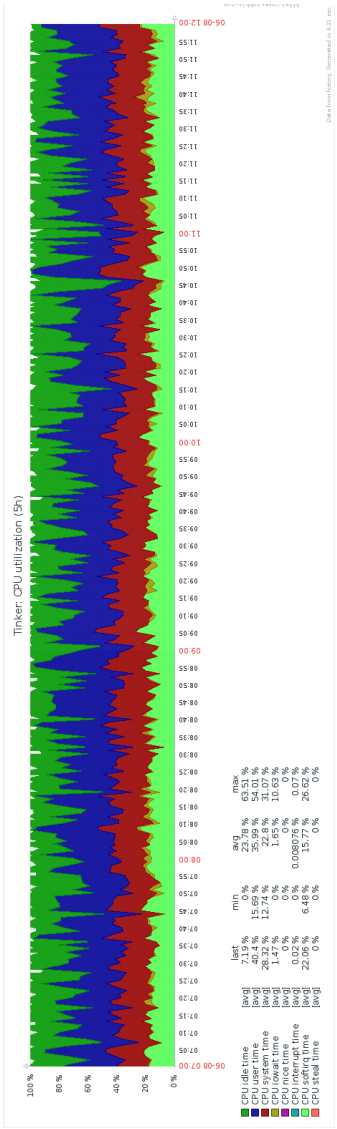

Abbildung 7.9: Prozentuale Lastverteilung auf der CPU von Tinker

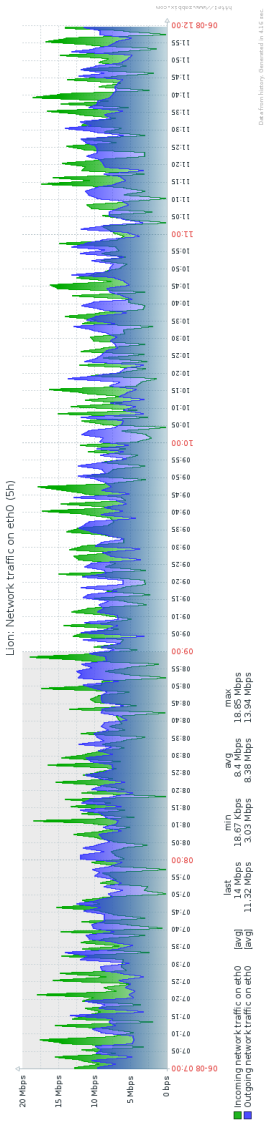

Abbildung 7.10: Traffic auf Eth0 bei 20 Megabyte auf Lion

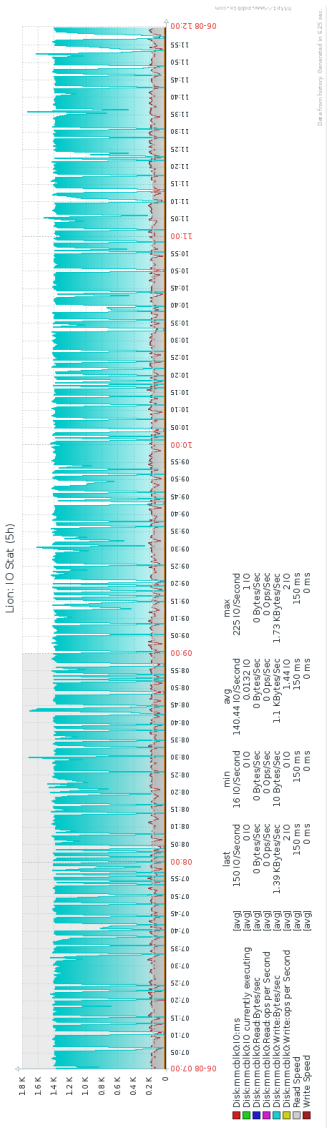

Abbildung 7.11: I/O Stats von Lion beim 20 Megabyte Netzwerk betrieb

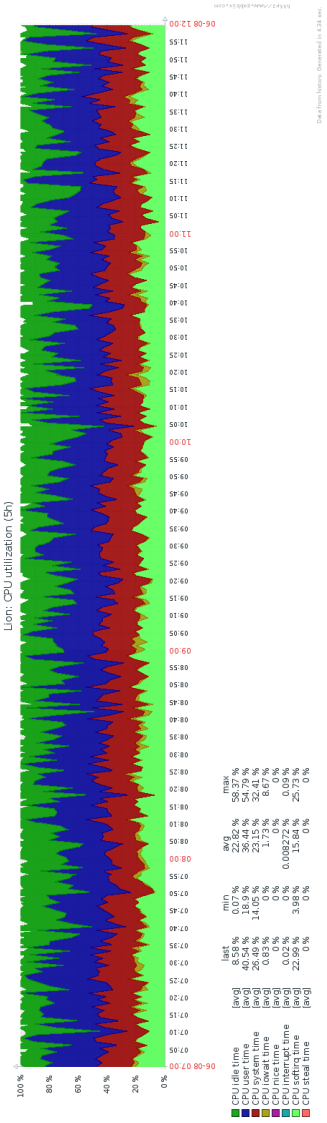

Abbildung 7.12: Prozentuale Lastverteilung auf der CPU von Lion

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.