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

Kapitel 1

Einführung

Netzwerke sind inzwischen fester Bestandteil der Informatik und finden überall Verwendung. Wenn früher Netzwerke nur im Militär, Universit"ten und Firmen benutzt wurden, haben auch normale Endverbraucher Zugriff auf Netzwerke, vorallem auf das Internet welches ein Netzwerk ist das über den ganzen Globus verteilt ist. Jedoch ist auch das Internet nur ein Netzwerk aus Netzwerken bricht man dies runter kommt man irgendwann in einem sogenannten Local Area Networks an. Inzwischen sind diese in fast jedem Haushalt zu finden, da diese es ermöglichen mehrere Computer über einen Router mit dem Internet zu verbinden. Dies ist jedoch nicht der einzige Zweck für die ein LAN verwendet wird. LANs ermöglichen einen Datenaustausch zwischen den im Netzwerk verbundenen Computern. 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 eigene Rechenzeit zu verfügung stellt. Eine weitere Anwendung für Verteilte Systeme stellt das Zabbix Framework dar, welches auch in dieser Bachelor Arbeit weiter betrachtet wird. 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. Mit der Hilfe von Zabbix und einigen selbstprogrammierten Bash Skripten möchte ich ein Beweisen das es in einem Netzwerk mit einer definierten Anzahl an homogenen Endgeräten möglich ist eine dem Optimale größe von Dateien die verschickt werden zu bestimmen. Dafür wurde ein Local Area Network bestehend aus vier Raspberry Pis aufgebaut. Diese wurden in einer Sterntopologie aufgebaut. Dieser wird standardmäig in Netzwerken verwendet. Da bei dieser Topologie der Ausfall von einem Computer keine Auswirkung auf den Rest des Netzwerks hat, leicht erweiterbar ist und leicht verständlich ist.[**book:CN2003**]. Dafür wurde 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 habe ich mir ein eigenes Testframework aufgebaut, welches es automati-

siert Dateien von den Computern verschickt und Informationen über die Dateien, Empfänger und benötigten Übermittlungszeiten speichert. Diese Logfiles werden mittels Textverarbeitung analysiert und in Tabellen dargestellt. Zabbix kann einige Informationen zu den Computern in Netz ausgeben. So ist es möglich mittels Zabbix die Prozessor Last, die Festplatten Last und die Last auf den Ethernet Ports der Raspberry Pis zu beobachten. Mit diesem Framework, welches in **??** nochmal genauer vorgestellt wird. Möchte ich beweisen das es möglich ist die optimale Paketgröße für ein Netzwerk zu bestimmen. Dazu werden mittels Textverarbeitung die Logfiles ausgewertet und die Daten die Werte Zabbix zur verfügung stellt in Tabellen aufbereitet. Dazu werden in dieser Arbeit drei verschiedene Testfälle betrachtet, kleine Dateien: Diese sind 20 Megabyte groß, mittlere Dateien: diese sind 200 Megabyte groß und große Dateien diese sind 2000 Megabyte groß. Am Ende dessen möchte ich die so gesammelten Ergebnisse vergleichen und überprüfen ob die Annahme, das es möglich ist eine optimale Paketgröße in einem Netzwerk zu bestimmen.

Kapitel 2

Das Framework

2.1 Verwendete Hardware

Die in diesem Framework verwendete Hardware sind folgende Geräte.

- 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 2.2 erklärt.

Als erstes soll ein Einblick in die Hardware Spezifikation der Raspberry Pis gegeben werden eingegeben werden. 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 2.2 erklärt.

Die beiden verwendeten Switches können in einem 100 oder 1000 Mbit Netz Arbeiten, jedoch sind die Raspberry Pis der Flaschenhals in diesem Aufbau und somit wird einem reinem 100 Mbit Netzwerk gearbeitet.

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	700 MHz	1024 MB	32 GB	100 MB/s

Tabelle 2.1: Spezifikation der Raspberry Pis

2.2 Aufbau der Software

Das Testframework besteht aus drei verschiedenen Teilen.

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

Die Eigentwicklungen sind alle in Bash programmiert, während Zabbix eine bereits fertige Open Source Lösung ist. In den folgenden Abschnitten werde ich einen Einblick in diese Teile geben.

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, welches wie Zabbix unter der GPL Lizenz vertrieben wird [LLC16]. Beide Systeme basieren auf einer Client-Server Architektur. Im weiteren wird jedoch nur Zabbix betrachtet. Zabbix besteht aus zwei Komponenten.

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. Eine genaueren Einblick in den Zabbix Server gibt es im Kapitel 3.

Zabbix Agent Die Clients, die im Netzwerk überwacht werden sollen, sind die sogenannten Agents, die an den Server die Informationen weiterleiten, die vom Server gefordert werden. In den späteren Kapiteln wird der Hauptfokus auf der I/O–Last der Festplatte und wie beim Server dem ein/- und ausgehenden Traffic auf dem Ethernet Port Eth0 liegen. Eine genaueren Einblick in den Zabbix Server gibt es im Kapitel 3.

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 die auf dem Server läuft, 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 mit dem Skript Get Logs die auf den Agents gelagerten Logfiles auf dem Rechner, der dieses Skript startet, gesammelt. Somit hat man die von den Endgeräten gesammelten Daten auf einem Rechner und kann mit der Weiterverarbeitung der Daten beginnen.

2. Zabbix Agent

Hintergrundrauschen: Diese Eigentwicklung 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 und baut auf der Secure Shell auf, so wird eine TCP Verbindung zum angesprochenen Host aufgebaut wird. So wird eine Last auf dem Netzwerk erzeugt, die mit Hilfe des Zabbix Servers gemessen werden kann. Außerdem speichert Hintergrundrauschen die Dauer, bis ein Paket erfolgreich bei seinem zufällig ausgewählten Empfänger angekommen ist. Es werden drei verschieden große Pakete verschickt 20 Megabyte, 200 Megabyte und 2 Gigabyte. Die Ergebnisse der Logfiles werden in Kapitel 4 betrachetet.

Startrauschen: wird automatisch auf den Agents auf den Endgeräten ausgeführt. Es dient als eine Zeitschaltuhr und ermöglicht ein zeitversetztes Starten des Scripts Hintergrundrauschen. Jedoch wird in den in dieser Ausarbeitung betrachtet Tests immer ein Zeitgleicher Start durchgeführt. Trotzdem 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: 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 dies jedoch weitergeleitet und in einer Schleife weiterverarbeitet so, das neben die Meldung die vom Befehl Ping Befehl kommt noch ein Datum vorgestellt kriegt. 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 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

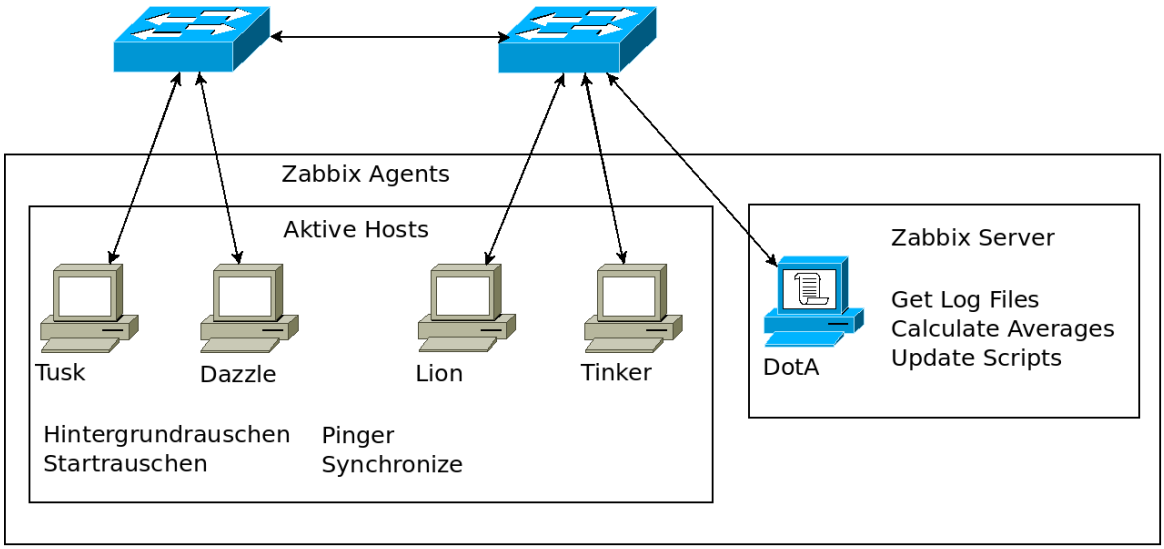

Abbildung 2.1: Aufbau des Netzwerks

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 In 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 Veriablendeklaration muss immer dem auszuführenden Test angepasst werden. Da die die Datei die verschickt wird über den Linux internen Befehl *duplicate data* geschrieben wird, muss man diesem einen Blockgröße und eine Anzahl an zu schreiben 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 dann mit der Auswertung beginnen.

Kapitel 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 Monitoring aufzuziehen mit mehreren Servern, diese sogenannten Proxies 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 Proxies Informationen zugeschickt. Die Aufgabe des Server 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 im Zabbix Server den Agents verschiedene Templates auflegen. Das in dem in Kapitel 2 gezeigte Framework verwendet primär das Template: *Template OS Linux*. Dieses Templates 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 in 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 selber sendet dann an den Server einen Request in dem der Agent nach den Daten fragt die der Server erwartet sind[Kra16]. Dies ist der Fall wenn vom Agent Daten abgefragt werden sollen die nicht vom Betriebssystem des Agents gesammelt werden. Das bedeutet das 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 der 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 benötigt, jedoch funktioniert PHP 7 noch nicht. 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. Jedoch ist der Server nicht im allgemeinen Prozess des in Kapitel 2 vorgestellten Frameworks tätig. Trotzdem wird in den folgenden Kapiteln der Zabbix Server nicht aussen vor gelassen und in die den Tests betrachtet werden, dabei werden hauptsächlich die von Zabbix Statistiken zur Perfomance und zum Traffic auf dem Ethernet Port des Servers genommen. Der Wert Perfomance 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 die verarbeiteten Werte pro Sekunde. Diese Werte lassen einen Schluss auf die Leistungsanforderungen des Servers schließen. Sollte die Queue einen bestimmten schwellwert erreichen wird auf dem Zabbix Frontend eine Warnung ausgegeben das der Server mit der Verarbeitung

nicht hinterherkommt. Dies kann soweit gehen das 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 betrachtet, 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 im Betriebssystem integrierte Monitoring Funktionen oder über die Zabbix eigene API und sendet diese je nach Art des Checks diese Informationen an den Agent. Da der Agent schon im Betriebssystem integrierte Funktionen benutzt ist dieser sehr effizient und verbraucht kaum spürbar die 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 das dieser weitesgehend ohne Administrator Rechte funktionieren kann. Da es auch möglich ist über einen Fernzugriff Agent Hostsysteme auszuschalten oder neuzustarten müssen für diese Funktionen dem Agent in der Rechteverwaltung diese 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 wären:

- 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 Konfigurations Dateien die Ports die der Server für das versenden von Requests benutzt eingestellt werden und auch die IP des Servers muss eingetragen werden. Da der Agent sonst eingehende Requests verwirft und somit die Sicherheit für den Host gewährleistet. In den Konfigurationsdateien der Agents ist es auch möglich die Active Checks zu notieren, was in diesem Versuchsaufbau auch gemacht wurde. Da das Template OS Linux keine Items hatte die das überwachen von der Festplatten Last hatte kann man dies über die Konfigurationsdateien einstellen. Das sieht in der Konfigurationsdatei dann so aus.

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 emphUserParameter= gibt an das 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 nachzuprüfen dazu liest es aus der Datei die in Unix Systemen unter /proc/-diskstats liegt, nach den zugehörigen Werten $1 und 4 sind Variablen die für die verarbeitung notwendig sind. Der in diesem Beispiel genannte UserParameter liest die Lese Operationen pro Sekunde aus der Datei und leitet diese an den Server weiter.

Kapitel 4

Versuche

Aufgrund der Unterschiede zwischen den verschiedenen Paketgrößen unterscheidet sich die Last auf den Hostsystemen, mit diesen drei Versuchen wird versucht die optimale größe zwischen den drei verschieden großen Paketen zu finden. Dazu werden die vom Zabbix Server und der selbstentwickelten Software Daten zur verfügung gestellten Daten mit mitteln der Stochastik analysiert, dabei wird der Durchschnitt $\varnothing$ und die Standardabweichung σ Betrachtet. Dabei werden folgende Werte betrachtet:

- Last auf dem Ethernet Port
- Festplattenlast
- CPU Last
- Anzahl der erfolgreich verschickten Pakete
- Menge der verschickten Byte

Die Ergebnisse aus Abschnitt 4.0.1, Abschnitt 4.0.2 und Abschnitt 4.0.3 werden in Kapitel 5 miteinander verglichen.

4.0.1 20 Megabyte Testlauf

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

Wie man in **??** sehen kann ist der durchschnittlich ausgehende Traffic auf dem Pi Dazzle 8,85 Mbit/s pro Sekunde. 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 ist durchschnittlich 19,205 Mbit/s. Der minimal eingehende Traffic, liegt bei durchschnittlich 28,45 Kbit/s. Die minimal Werte entstehen, wenn der Pi kein Pakete empfängt und nur noch mit dem Zabbix Server kommuniziert. Dies wird man in die folgenden Tests auch so beobachten können. Die Standardabweichung der durchschnittlichen und der Maximallast sind geringer als

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 MB 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	154,09 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	120,825 Ops/s
Agent σ	0,028722813 KB/s	0 KB/s	45,4928 Ops/s

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

1 Mbit/s. Aus dieser geringen Abweichung vom Durchschnitt kann man schließen das die Hosts gleichmäßig verteilt Pakete empfangen.

Da, aber auch alle Hosts nicht nur Empfänger sondern auch gleichzeitig Sender sind wird auch der ausgehende Datenstrom betrachtet. Aus der Tabelle 4.2 kann man ablesen das 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 der vom eingehenden Traffic. Jedoch bei der minimallast sieht man das der Host Lion einen Wert von 3,03 Mbit/s aufweist. Dadurch erhöht sich der Durchschnitt der Minimallast drastisch. Man kann auch beobachten das Durchschnitt des Maxmimal Ausgehenden Traffics 4 Megabit/s geringer ist als der des eingehenden.

In der **??** sieht man das die Festplatte des Raspberry Pis konstant beschrieben wird, lese Operationen finden überhaupt nicht statt. Im durchschnitt werden 1.07 Kilobytes die Sekunde

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

geschrieben. Was nicht annährend die Maximale Schreibgeschwindigkeit der verwendeten SanDisk SD Karten ist, welche bei 30 Megabyte pro Sekunde [Cor16] liegt. Aus der Tabelle 4.3 lässt sich also schließen das die Festplatten der Agents kaum belastet werden.

Betrachtet man nun die Tabelle 4.4 spiegelen sich die Ergebnisse aus der Tabelle 4.3 wieder. Die Zeit die die CPU braucht um auf den Abschluss einer Lese oder Schreib Operation zu warten hält sich sehr gering. So werden durchschnittlich 1,26 % der CPU Zeit für Lese und Schreib Operationen verwendet. Der Großteil der CPU Zeit wird bei allen Hosts, dafür verwendet die User Anwendungen laufen zu lassen, im Schnitt 36,14 %. Idle gibt an wie viel der % vom Prozessor ohne eine Aufgabe war. Während die System Time für die verwaltung von der Netzwerkaufgaben zuständig war, wie das versenden und das empfangen von den Paketen die im Netzwerk verschickt werden. *An Julian: Für was ist die Software IRQ time ? Durch was werden die ganzen Software Interrupt ausgelöst ich peils nicht.*Wenn man nun die Werte der Standardabweichung der CPU betrachtet bestägtigt sich die Annahme die man aus der Tabelle 4.2 und der Tabelle 4.1 gezogen hatte, nämlich das alle Hosts gleichmäßig belastet worden sind. Die Standardbweichung der Erwartungswerte geht bei allen Werten nicht über 1 %, welches ein Indikator dafür ist das die Prozessoren auf den Hosts in etwa die selbe Last tragen.

Es wurden über eine dauer von 12 Stunden 10937 Pakete im Netzwerk verschickt. Die Menge der Daten die verschickt wurden sind 213,61 Gigabyte in diesem Netzwerk . Rechnet man dies auf eine Stunde runter kommt man auf 17,8 Gigabyte pro Stunde oder 911,41 Pakete pro Stunde. Wieder lässt sich eine gute gleichverteilung erkennen. Die Standardabweichung der Erfolgreich verschickten Pakete liegt 15 Paketen und auch die Fehlerrate ist sehr gering, ein Host hat sogar innerhalb von 12 Stunden kein einziges Paket nicht erfolgreich absenden können. Generell ist eine sehr geringe Menge an Daten verloren gegangen, insgesamt haben 80 Megabyte ihr Ziel nicht Ordnungsgemäß erreicht.

4.0.2 200 Megabyte Testlauf

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

Wie man in **??** sehen kann ist der durchschnittlich ausgehende Traffic auf dem Pi Dazzle

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 gesendet	0,01427 %	0,308004 GB

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

8,85 Mbit/s pro Sekunde. Der eingehende Traffic beträgt durchschnittlich 8,13 Mbit/s. In der Tabelle 4.6 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 minimal Werte entstehen, wenn der Pi kein Pakete empfängt und nur noch mit dem Zabbix Server kommuniziert. Dies wird man in die folgenden Tests auch so beobachten können. Die Standardabweichung der durchschnittlichen und der Maximallast sind geringer als 1 Mbit/s. Aus dieser geringen Abweichung vom Durchschnitt kann man schließen das die Hosts gleichmäßig verteilt Pakete empfangen.

Da, aber auch alle Hosts nicht nur Empfänger sondern auch gleichzeitig Sender sind wird auch der ausgehende Datenstrom betrachtet. Aus der Tabelle 4.7 kann man ablesen das 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 der vom eingehenden Traffic. Jedoch bei der minimallast sieht man das der Host Lion einen Wert von 3,03 Mbit/s aufweist. Dadurch erhöht sich der Durchschnitt der Minimallast drastisch. Man kann auch beobachten das Durchschnitt des Maxmimal Ausgehenden Traffics 4 Megabit/s geringer ist als der des eingehenden.

In der **??** sieht man das die Festplatte des Raspberry Pis konstant beschrieben wird, lese Operationen finden überhaupt nicht statt. Im 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 4.8 lässt sich also schließen das die Festplatten der Agents kaum belastet werden.

Betrachtet man nun die Tabelle 4.9 spiegelen sich die Ergebnisse aus der Tabelle 4.8 wieder. Die Zeit die die CPU braucht um auf den Abschluss einer Lese oder Schreib Operation zu warten hält sich sehr gering. So werden durchschnittlich 1,26 % der CPU Zeit für Lese und Schreib Operationen verwendet. Der Großteil der CPU Zeit wird bei allen Hosts, dafür verwendet die User Anwendungen laufen zu lassen, im Schnitt 36,14 %. Idle gibt an wie viel der

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 4.6: Eingehender Traffic auf den Ethernet Ports bei 20 MB 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 4.7: 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	154,09 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	120,825 Ops/s
Agent σ	0,028722813 KB/s	0 KB/s	45,4928 Ops/s

Tabelle 4.8: 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 4.9: 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 gesendet	0,01427 %	0,308004 GB

Tabelle 4.10: Anzahl der gesendeten Pakete über einen Zeitraum von 12 Stunden

% vom Prozessor ohne eine Aufgabe war. Während die System Time für die verwaltung von der Netzwerkaufgaben zuständig war, wie das versenden und das empfangen von den Paketen die im Netzwerk verschickt werden. *An Julian: Für was ist die Software IRQ time ? Durch was werden die ganzen Software Interrupt ausgelöst ich peils nicht.*Wenn man nun die Werte der Standardabweichung der CPU betrachtet bestägtigt sich die Annahme die man aus der Tabelle 4.7 und der Tabelle 4.6 gezogen hatte, nämlich das alle Hosts gleichmäßig belastet worden sind. Die Standardbweichung der Erwartungswerte geht bei allen Werten nicht über 1 %, welches ein Indikator dafür ist das die Prozessoren auf den Hosts in etwa die selbe Last tragen.

Es wurden über eine dauer von 12 Stunden 10937 Pakete im Netzwerk verschickt. Die Menge der Daten die verschickt wurden sind 213,61 Gigabyte in diesem Netzwerk . Rechnet man dies auf eine Stunde runter kommt man auf 17,8 Gigabyte pro Stunde oder 911,41 Pakete pro Stunde. Wieder lässt sich eine gute gleichverteilung erkennen. Die Standardabweichung der Erfolgreich verschickten Pakete liegt 15 Paketen und auch die Fehlerrate ist sehr gering, ein Host hat sogar innerhalb von 12 Stunden kein einziges Paket nicht erfolgreich absenden können. Generell ist eine sehr geringe Menge an Daten verloren gegangen, insgesamt haben 80 Megabyte ihr Ziel nicht Ordnungsgemäß erreicht.

4.0.3 2 Gigabyte Testlauf

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

Wie man in **??** sehen kann ist der durchschnittlich eingehende Durchsatz auf dem Pi Dazzle 8,04 Megabit pro Sekunde. Der ausgehende Traffic beträgt durchschnittlich 8,62 Megabit pro Sekunde. Rechnet man diese Werte in Bytes pro Sekunde um beträgt der ausgehende Datenstrom 1,0775 Megabyte/s und der durchschnittlich eingehende Datenstrom 1,005 Megabyte/s Daraus können wir schließen das der Ethernet Port von Dazzle unter einer durschnittlichen I/O-Last von 2,0825 Megabyte/s stand. Woraus folgt das der Ethernet Port zu 0,20825% belastet ist.

In der **??** sind die Durchnittswerte für den Traffic der jeweiligen Agents eingespeichert. Auch

Agent	Minimal Kb/s	Durchschnitt Mb/s	Maximal Mb/s
DotA	7,46 Kb/s	0,00792 Mb/s	0,01195 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	7,64 Kb/s	8,63 Mb/s	24,98 Mb/s
Agent ∅	6,145 Kb/s	7,975 Mb/s	23,9475 Mb/s
Agent & Server ∅	6,408 Kb/s	6,381584 Mb/s	19,708016411 Mb/s

Tabelle 4.11: Eingehender Traffic auf den Ethernet Ports bei 20 MB Paketen auf allen Pis

Agent	Minimal Kb/s	Durchschnitt Mb/s	Maximal Mb/s
DotA	11,58 Kb/s	0,01514 Mb/s	0,08772 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	9,2 Mb/s	24,97 Mb/s
Agent ∅	1035,35 Kb/s	8,165 Mb/s	23,09 Mb/s
Agent & Server ∅	830,582 Kb/s	6,535028 Mb/s	18,489544 Mb/s

Tabelle 4.12: Ausgehender Traffic auf den Ethernet Ports bei 20 MB Paketen auf allen Pis

der des Servers, da dieser im selben Netzwerk aufgestellt ist wie die anderen Agents. Die Tabelle zeigt uns auch das sich die Agents alle in einem ähnlichen Umfeld was deren Last angeht. Die Standardabweichung σ der Agents bestätigt diese Annahme diese liegt nach **??** bei 0,15411 Megabit.

In der **??** sieht man das die Festplatte des Raspberry Pis konstant beschrieben wird, lese Operationen finden überhaupt nicht statt. Im durchschnitt werden 1.04 Kilobytes die Sekunde geschrieben. Mit einem einer Maximallast von 1.86 Kilobytes die Sekunde. Was nicht annährend die Maximale Schreibgeschwindigkeit der verwendeten SanDisk SD Karten ist welche bei 30 Megabyte pro Sekunde [Cor16] liegen.

Agent	Minimal Kb/s	Durchschnitt Mb/s	Max Mb/s
Agents σ	0,015 Kb/s	2,5868175429 Mb/s	1,7957919562 Mb/s
Agents & Server σ	0,5261710748 Kb/s	3,9381719358 Mb/s	9,7080164611 Mb/s

Tabelle 4.13: Standardabweichung der Eingehende der Last auf dem Ethernet Port bei 20 MB großen Paketen

Agent	Minimal Kb/s	Durchschnitt Mb/s	Max Mb/s
Agents σ	0,0294745653 Kb/s	1,6381773408 Mb/s	2,191540554 Mb/s
Agents & Server σ	1,5672217456 Kb/s	3,5740921761 Mb/s	9,4073939873 Mb/s

Tabelle 4.14: Standardabweichung der Ausgehenden der Last auf dem Ethernet Port bei 20 MB großen Paketen

Agent	Schreiben KB/s	Input/Output Ops/s
Dazzle	1,04 KB/s	154,09 Ops/s
Tusk	1,11 KB/s	42,4 Ops/s
Tinker	1,07 KB/s	142,5 Ops/s
Lion	1,04 KB/s	144,31 Ops/s
Agent ∅	1,065 KB/s	120,825 Ops/s
Agent σ	0,028722813 KB/s	45,4928 Ops/s

Tabelle 4.15: 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 %	1,73 %	15,81 %
Agent σ	0,10 %	0,97 %	0,46 %	0,61 %	0,20 %

Tabelle 4.16: CPU Last Verteilung

Kapitel 5

Vergleich VM/HW

5.1 Versuchsergebnisse

5.2 Kosten nutzen Faktor

Kapitel 6

Fazit

Literatur

[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.

[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/.

[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/.