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 **iv**

Tabellenverzeichnis **v**

1 Einführung **1**

2 Das Framework **2**

2.1 Verwendete Hardware 2

2.2 Aufbau der Software 2

2.2.1 Zabbix 3

2.2.2 Eigenentwicklung 3

2.3 Einsatz im Netzwerk 4

3 Zabbix **6**

3.1 Zabbix Server 6

3.2 Zabbix Agent 8

4 Versuche **9**

4.1 Raspberry Pi Versuche 9

4.1.1 Kein Traffic 9

4.1.2 Normalbetrieb 10

4.1.3 Loop 12

4.1.4 Doppelte IP 14

4.2 Virtual Machine Versuche 15

4.2.1 Gemachte Tests 15

4.2.2 Ergebnisse 15

5 Vergleich VM/HW **16**

5.1 Versuchsergebnisse 16

5.2 Kosten nutzen Faktor 16

6 Fazit **17**

Abbildungsverzeichnis

2.1 Aufbau des Netzwerks 5

4.1 Traffic auf Eth0 beim Zabbix Server bei keinem Datenaustausch 9
4.2 Traffic auf Eth0 bei Normalbetrieb auf Dazzle 11
4.3 I/O Stats von Dazzle beim Normalbetrieb 12
4.4 Traffic auf Eth0 bei einem Loop 13
4.5 I/O-Last auf der Festplatte bei einem Loop 13
4.6 Fehlermeldung auf dem Zabbix Dashboard bezüglich Tinker 14

Tabellenverzeichnis

2.1 Spezifikation der Raspberry Pis 2

4.1 I/O Zeiten bei keinem Datenaustausch auf den Pis 10
4.2 Traffic bei keinem Datenaustausch auf den Pis 10
4.3 Standarbweichung bei keinem Datenaustausch 10
4.4 Normalbetrieb Traffic auf allen Pis 11
4.5 Normalbetrieb Standarbweichung der Werte 11
4.6 I/O Zeiten bei Normalbetrieb auf den Pis 12
4.7 Traffic Durchschnittswerte bei Doppelt belegter IP 14
4.8 Doppelte IP Standarbweichung der Werte 14

Listingverzeichnis

Kapitel 1

Einführung

Hallo sehr geschätzter Leser Willkommen bei meiner Bachelor Thesis in der ich den die Differenz zwischen Virtuellen Maschinen und realen Maschinen in einem Netzwerk vergleichen möchte. Dazu habe ich selber ein Netzwerktestframework entwickelt und von Netzwerktestframeworks demonstrieren, die in Netzwerken die aus realen Maschinen bestehen und aus virtuellen Maschinen verwendet werden. Wieso hat das eine relevanz? Das werde ich euch erklären wieso. Wir leben in einer immer weiter hochvernetzten Welt sind und die Industrie 4.0 wird immer mehr bestandteil unserer Welt, Kühlschränke werden mit dem Internet verbunden. Um eine hohe netzstabilität gewährleisten müssen wir.

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 eigenständigend 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.

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.

2.2 Aufbau der Software

Das Testframework besteht aus drei verschiedenen Teilen.

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

- Zabbix
- Eigentwicklung auf dem Server
- Eigentwicklung 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, welches seit 2005 von der Firma Zabbix SIA weiterentwickelt wird [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 dessen Weiterverarbeitung betreiben.

Calculate Average: Berechnet den Durschnitt der Dauer, die ein Paket braucht, um erfolgreich versendet zu werden.

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. So wird eine Last auf dem Netzwerk erzeugt, die Mithilfe 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 werden in Kapitel 4 erörtert.

Startrauschen: wird automatisch zum Start eines der Endgeräte ausgeführt. Es dient als eine Zeitschaltuhr und ermöglicht ein zeitversetztes Starten des Scripts Hintergrundrauschen, wodurch man den sequentiellen Anstieg an Last im Netzwerk beobachten kann.

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 Servers.

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

2.3 Einsatz im Netzwerk

Mit dem Einsatz der im vorherigen Abschnitt vorgestellten Software ist das Testframework aufgebaut. In der 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.

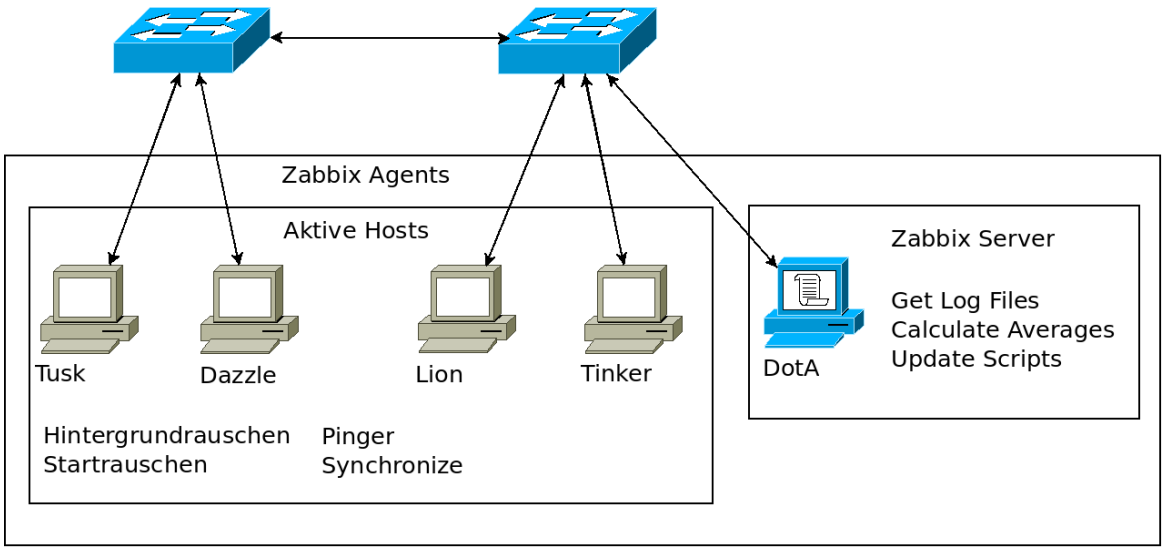

Abbildung 2.1: Aufbau des Netzwerks

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 Zabbix Framework. 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 erstellt werden. Die dazu benötigten Daten ist die Agent IP und ein eindeutiger Name. Wenn nun der Agent erstellt worden ist kann man über die Konfiguration 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 eingestellten Agent anfordern. Ein Passive Check geschieht über einen Request diese sind in JSON verfasst 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

[SIA16b]. 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 die zu sammelnden Daten enthalten 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)

[SIA16c] Der Zabbix Server selber ist auch als ein Agent konfiguriert, der sich selber überwacht. Jedoch ist der Server nicht im allgemeinen Prozess des in **??** 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 Leistung 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 nicht aktuell oder sogar 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 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 ohne Administrator Rechte funktioniert, 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 eine Abwandlung von Debian Jessie ist.

Kapitel 4

Versuche

Aufgrund der Unterschiede zwischen einer Virtuellen und realen Maschine können nicht auf beiden Systemen die gleichen Versuche gemacht. Deshalb müssen von den oben vorgestellten Versuchen ein paar Versuche entfernt werden da diese in einem realen Netzwerk nicht auftreten. Bei allen Versuchen werden die Daten die vom Zabbix Server zur verfügung gestellt werden mit mitteln der Stochastik analysiert, vorallem wird ein Fokus auf die Standardabweichung des Traffics im Netzwerk gelegt. Als auch auf die Werte die man von der Festplatte sammeln kann. So ist es möglich die Werte untereinander zu vergleichen. Mit den Ergebnissen aus diesem Kapitel werden in Kapitel 5 die Ergebnisse aus Abschnitt 4.1 und Abschnitt 4.2 verglichen.

4.1 Raspberry Pi Versuche

4.1.1 Kein Traffic

In diesem Versuch wird der Normale Netzaufbau benutzt. Jedoch wird als Besonderheit das Skript Hintergrundrauschen nicht ausgeführt. So erhält man ein Gefühl dafür wie sich das Netzwerk verhält wenn das Netzwerk nicht in Betrieb ist. Für diesen Test wird als erstes der Zabbix Server betrachtet. Dies ist vorallem interessant, da auch Zabbix selber Traffic erzeugt und um sich weiter mit dem Netzwerk ausseinander zu setzen ist es wichtig zu wissen was im Netzwerk passiert wenn auf ihm keine Last liegt.

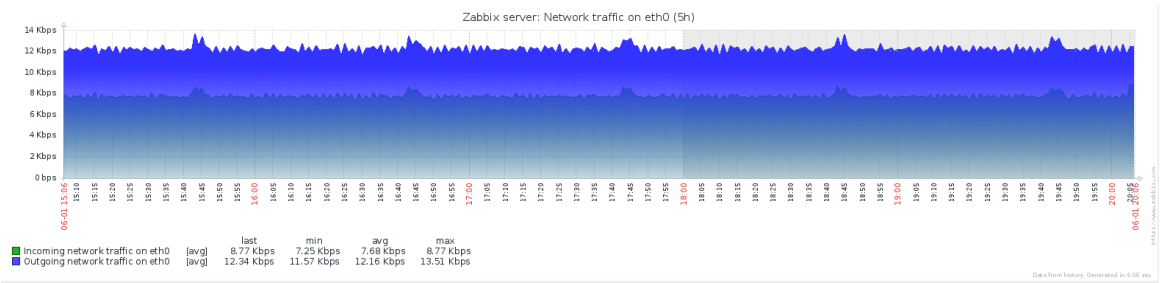

Abbildung 4.1: Traffic auf Eth0 beim Zabbix Server bei keinem Datenaustausch

Die Tabelle 4.2 zeigt die durchschnittlichen Übertragungswerte der einzelnen Pis auf, wie

Agent	Schreiben B/s	Lesen B/s
Dazzle	0,710000 B/s	0 B/s
Tusk	0,183600 B/s	0 B/s
Tinker	0,693300 B/s	0 B/s
Lion	0,745000 B/s	0 B/s
∅ Agent	0,582975 B/s	0 B/s
Standardabweichung	0,231333 B/s	0 B/s

Tabelle 4.1: I/O Zeiten bei keinem Datenaustausch auf den Pis

Agent	Eingehende Kb/s	Ausgehende Kb/s	Gesamt Kb/s	Auslastung von Eth0 %
DotA	7,68 Kb/s	12,16 Kb/s	19,84 Kb/s	0,01984 %
Dazzle	2,19 Kb/s	2,72 Kb/s	4,91 Kb/s	0,00490 %
Tusk	2,23 Kb/s	2,67 Kb/s	4,9 Kb/s	0,00491 %
Tinker	2,19 Kb/s	2,72 Kb/s	4,91 Kb/s	0,00491 %
Lion	2,19 Kb/s	2,72 Kb/s	4,91 Kb/s	0,00491 %
∅ Agent	2,20 Kb/s	2,37 Kb/s	4,9075 Kb/s	0.0049075 %
∅ Agent & Server	3,30 Kb/s	4,33 Kb/s	7,894 Kb/s	0.007894 %

Tabelle 4.2: Traffic bei keinem Datenaustausch auf den Pis

Agent	Eingehende Kb/s	Ausgehende Kb/s	Gesamt Kb/s	Last auf Eth0 %
Agents	0,034641 Kb/s	0,043301 Kb/s	0,0086600 Kb/s	0,000019625 %
Agents & Server	4,903183 Kb/s	8,995337 Kb/s	13,35604 Kb/s	0,089822765 %

Tabelle 4.3: Standarbweichung bei keinem Datenaustausch

man sieht befindet sich der Durchschnitt im geringen Kilobit pro Sekunde Bereich. Es ist davon auszugehen das der gesamte sichtbare Traffic durch den Zabbix Server zustande kommt. Auch die I-O Last auf den Pis zeigt das kaum schreib operationen geschehen, Lese Operationen geschehen bei den Pis gar nicht. In der Zabbix queue stapeln sich keine Daten, im Durschnitt verarbeitet der Zabbix 4,05 Werte pro Sekunde.

4.1.2 Normalbetrieb

Der erste durchgeführte Test ist der Versuch Normalbetrieb. In diesem Test ist das Ziel zu sehen wie sich das Netzwerk verhält wenn keine Störungen stattfinden. Dadurch erhalten wir einen Nennwert mit es möglich ist Aussagen über das Netzwerk zu treffen wenn ein Störungsfall eintritt. Dieser Test wird über einen Zeitraum von fünf Stunden absolviert. Die Dateien die im Netzwerk verschickt werden sind alle 20 Megabytes groß, dadurch wird ein gleichmäßiger Datenstrom erzeugt. Als erste Kontroll Instanz werden die Daten die vom Zabbix Server erzeugt werden überprüft, da dieses Tool ein für den Endverbraucher praktisches Frontend besitzt. Wie sich ein Agent verhält wenn im Netz Traffic Störungsfrei läuft sieht man in den folgenden zwei

Agent	Eingehende Mb/s	Ausgehende Mb/s	Gesamt Mb/s	Auslastung von Eth0 %
DotA	0,0769 Mb/s	0,1222 Mb/s	0,19900 Mb/s	0,0199 %
Dazzle	8,04 Mb/s	8,62 Mb/s	16,66 Mb/s	1,666 %
Tusk	8,48 Mb/s	8,37 Mb/s	16,85 Mb/s	1,685 %
Tinker	8,42 Mb/s	8,56 Mb/s	16,98 Mb/s	1,698 %
Lion	8,42 Mb/s	8,65 Mb/s	16,89 Mb/s	1,707 %
∅ Agent	8,34 Mb/s	8,55 Mb/s	16,89 Mb/s	1,689 %
∅ Agent & Server	6,68736 Mb/s	6,86444 Mb/s	13,5518 Mb/s	1,35518 %

Tabelle 4.4: Normalbetrieb Traffic auf allen Pis

Agent	Eingehende Mb/s	Ausgehende Mb/s	Gesamt Mb/s	Last auf Eth0 %
Agents	0,174929 Mb/s	0,108857 Mb/s	0,15411 Mb/s	0,01541 %
Agents & Server	3,308981 Mb/s	3,372526 Mb/s	6,67782 Mb/s	0,66782 %

Tabelle 4.5: Normalbetrieb Standarbweichung der Werte

Graphen.

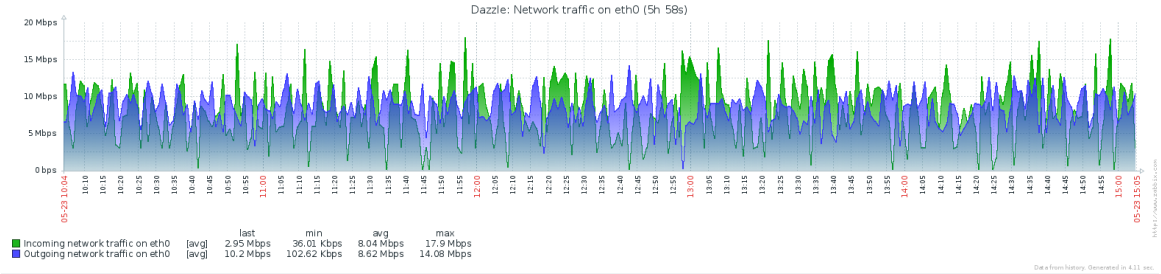

Abbildung 4.2: Traffic auf Eth0 bei Normalbetrieb auf Dazzle

Wie man in Abb. 4.2 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 20,825% belastet war.

In der Tabelle 4.4 sind die Durchnittswerte für den Traffic der jeweiligen Agents eingespeichert. Auch 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 befinden was deren Input sowie Output betrifft. So beträgt die Standardabweichung der Pis nach der Tabelle 4.5

In der Abb. 4.3 sieht man das die Festplatte des Raspberry Pis konstant beschrieben wird. Im durchschnitt werden 1.04 Kilobytes die Sekunde geschrieben. Mit einem einer Maximallast von 1.86 Kilobytes die Sekunde. Was nicht annähernd die Maximale Schreibgeschwindigkeit der

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
Standardabweichung	0,028722813 KB/s	45,4928 Ops/s

Tabelle 4.6: I/O Zeiten bei Normalbetrieb auf den Pis

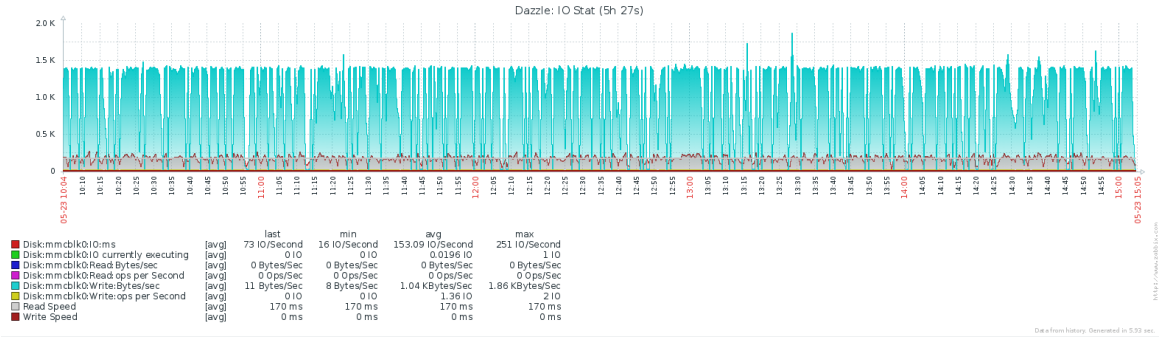

Abbildung 4.3: I/O Stats von Dazzle beim Normalbetrieb

verwendeten SanDisk SD Karten ist welche bei 30 Megabyte[Cor16] pro Sekunde liegen.

Schlussfolgerung Test: Normalbetrieb

Nach erfolgreichem durchführen des Tests kann man sehen wie sich die einzelnen Endgeräte im Netzwerk verhalten wenn Pakete von 20 MB größe darüber läuft. Man kann sehen das der Datenstrom konstant bleibt und es keine hohen Abweichungen im eingehenden und ausgehenden Traffic gibt. Woraus man schließen kann das ein Reibungsloser Ablauf im Netzwerk geleistet ist. Jedoch sieht man das Tusk sehr wenige I/O Operationen pro Sekunde vornimmt. Um das genauer beurteilen zu können müsste man sich angucken wie die CPU Zeiten genutzt werden. Jedoch ist zu bedenken das die Metrekt der I/O Operationen pro Sekunde

4.1.3 Loop

Wie in **??** beschrieben ist ein Loop ein Fehler der auftritt wenn in einem Switch zwei Ports miteinander verbunden worden sind. Dasselbe gilt auch wenn zwei Switches miteinander verbunden werden. Loops sind Fehler die auftreten wenn der Endnutzer nicht mit dem Umgang der Hardware vertraut ist, oder die IT-Infrastruktur zu groß wird. Der erwartete Ausgang des Tests ist nach Abschnitt 3.8 [Sch14] ein totaler Ausfall des Netzwerkes. Die nun gezeigten Ergebnisse bestätigen diese Annahme.

Wie man in der Graphik sehen kann bricht die Verbindung zum Zabbix Server komplett ab. Es findet nichtmal der wie in Abschnitt 4.1.1 aufgezeigte Server Poll statt, der einen geringen

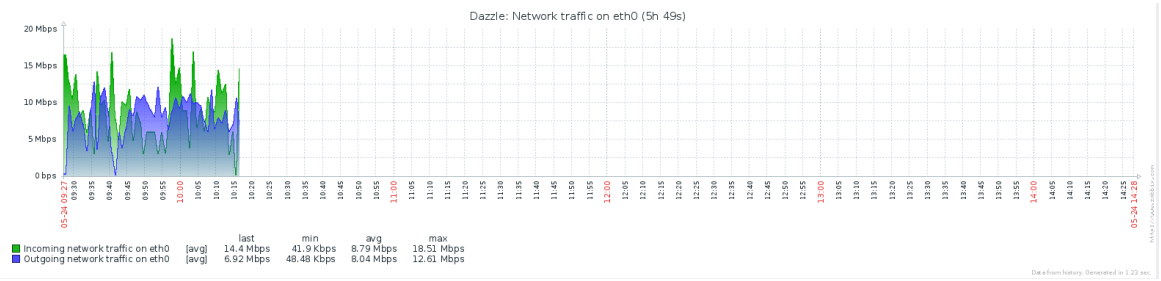

Abbildung 4.4: Traffic auf Eth0 bei einem Loop

Traffic erzeugt. Auch der Graph für die Festplatten Last bricht zum Zeitpunkt des Loops ab. Wieder werden im Graphen auch die Durchschnittswerte des Traffics aufgezeigt. Wenn man nun die Werte mit denen aus dem Abschnitt 4.1.2 vergleicht stellt man fest das die Durchschnittswerte eine geringe Abweichung aufzeigen. Dies liegt dem zugrunde das Zabbix keine neue Daten von den Agents erhalten kann. Da die Berechnung des durchschnittlichen Traffic erst dann erfolgt wenn der Agent eine Naricht über seinen Zustand versendet. Da diese jedoch nicht den Server erreichen bleiben die Durchschnnittswerte wie beim Normalbetrieb.

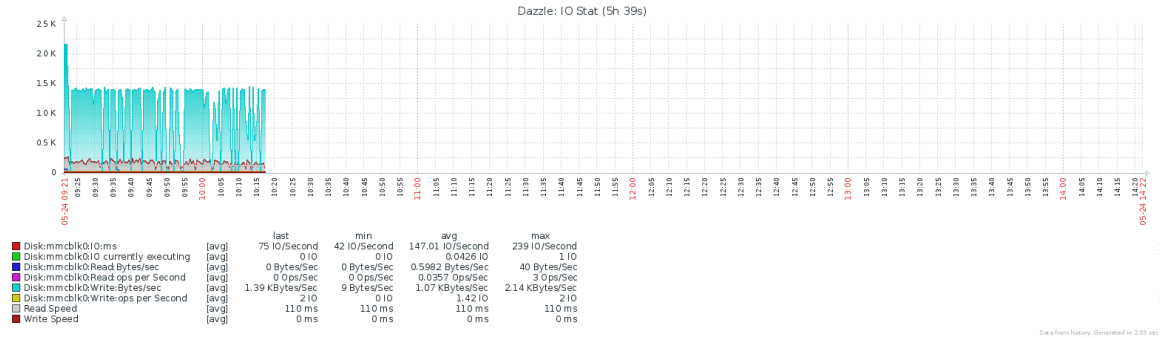

Abbildung 4.5: I/O-Last auf der Festplatte bei einem Loop

Hier kann man sehen das der Loop auch die Informationen die der Agent über die Festplatte verschickt unerreichbar sind. Dies hat jedoch aus Erfahrungswerten keinen Einfluß auf die Performanz der Agents. Man kann davon ausgehen das sich die I/O-Last der Agents auf den Wert der in Abschnitt 4.1.1 gezeigten Werte einpendelt.

Schlussfolgerung Test: Loop

Wie man in Abschnitt 4.1.3 sehen kann, zeichnet sich ein Loop dadurch aus der komplette Traffic in einem Netzwerk zusammenbricht. In Zabbix selber zeichnet sich das dadurch aus das die Graphen einen direkten Schnitt aufzeigen und auch über einen längeren Zeitraum sich die Durchschnittswerte die von den Agents verschickt werden nicht verändern. Auch die Logfiles geben aufschluss über die Transportierten Pakete so kann man aus diesen lesen das nur noch die Pakete die an den Lokalen Host geschickt werden ihr Ziel erreichen.

Agent	Eingehende Mb/s	Ausgehende Mb/s	Gesamt Mb/s	Auslastung von Eth0 %
DotA	0,00785 Mb/s	0,0924 Mb/s	0,1709 Mb/s	0,01709 %
Dazzle	20,15 Mb/s	10,32 Mb/s	30,47 Mb/s	1,809 %
Tusk	7,7 Mb/s	10,32 Mb/s	18,09 Mb/s	1,809 %
Lion	6,72 Mb/s	6,19 Mb/s	12,91 Mb/s	1,291 %
∅ Agent	11,55 Mb/s	8,9433 Mb/s	20,49 Mb/s	2,049 %
∅ Agent & Server	8,679625 Mb/s	6,7306 Mb/s	15,410225 Mb/s	1,5410255 %

Tabelle 4.7: Traffic Durchschnittswerte bei Doppelt belegter IP

Agent	Eingehende Mb/s	Ausgehende Mb/s	Gesamt Mb/s	Last auf Eth0 %
Agents	6,908559 Mb/s	1,946900 Mb/s	7,36697 Mb/s	0,73670 %
Agents & Server	7,249420 Mb/s	4,187049 Mb/s	10,8681 Mb/s	1,08681 %

Tabelle 4.8: Doppelte IP Standarbweichung der Werte

4.1.4 Doppelte IP

In diesem Test wurde dem Agent Tinker dieselbe IP vergeben wie sie Dazzle hat. Dadurch entsteht ein Netzwerkadressenkonflikt. Das erste was passiert ist das die Pakete die für den Pi Tinker bestimmt sind ihr Ziel nichtmehr erreichen können.

In diesem Test wird der Raspberry Pi Tinker ignoriert da dieser in dem Test keine Pakete erhalten hat und auch vom Zabbix Server nicht auffindbar ist wie die Abb. 4.6 nahelegt.

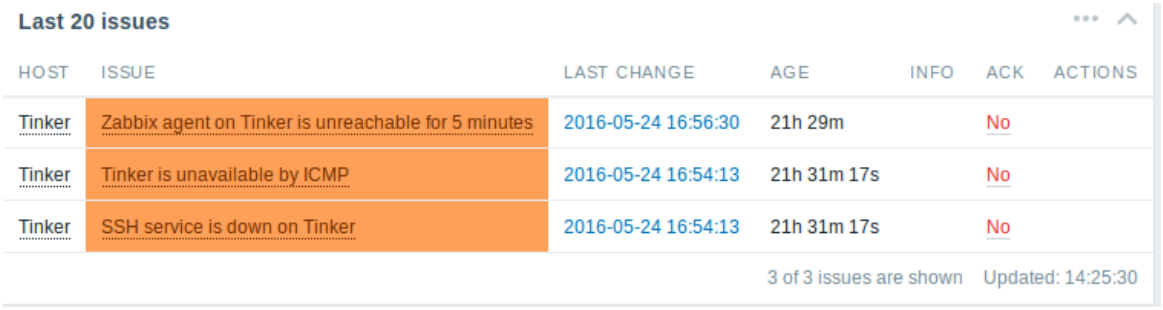

Abbildung 4.6: Fehlermeldung auf dem Zabbix Dashboard bezüglich Tinker

Wie man der Tabelle 4.7 sehen kann wird Dazzle mehr belastet als sonst. Vergleicht man die Standarbweichung von Tabelle 4.8 mit der Tabelle 4.5 sieht man das die Abweichung sechsfach größer ist. Der Eingehende Datenstrom ist doppelt so hoch wie beim Normalbetrieb. Auch der Ausgehende Traffic ist bei Dazzle gestiegen, jedoch blieb der Durchschnitt ungefähr der gleiche.

4.2 Virtual Machine Versuche

4.2.1 Gemachte Tests

4.2.2 Ergebnisse

Kapitel 5

Vergleich VM/HW

5.1 Versuchsergebnisse

5.2 Kosten nutzen Faktor

Kapitel 6

Fazit

Literatur

[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/.

[Sch14] Rüdiger Schreiner. *Computernetzwerke : von den Grundlagen zur Funktion und Anwendung*. 5., erw. Aufl. München: Hanser, 2014. ISBN: 9783446441545. URL: http://www.hanser-elibrary.com/doi/book/10.3139/9783446441545.

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

[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/.