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

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 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.

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
- 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. 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 werden in Kapitel 4 erörtert.

Startrauschen: wird automatisch auf den Agents auf den Endgeräten ausgeführt. Es dient als eine Zeitschaltuhr und ermöglicht ein zeitversetztes Starten des Scripts Hintergrundrauschen.

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 herum 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 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.

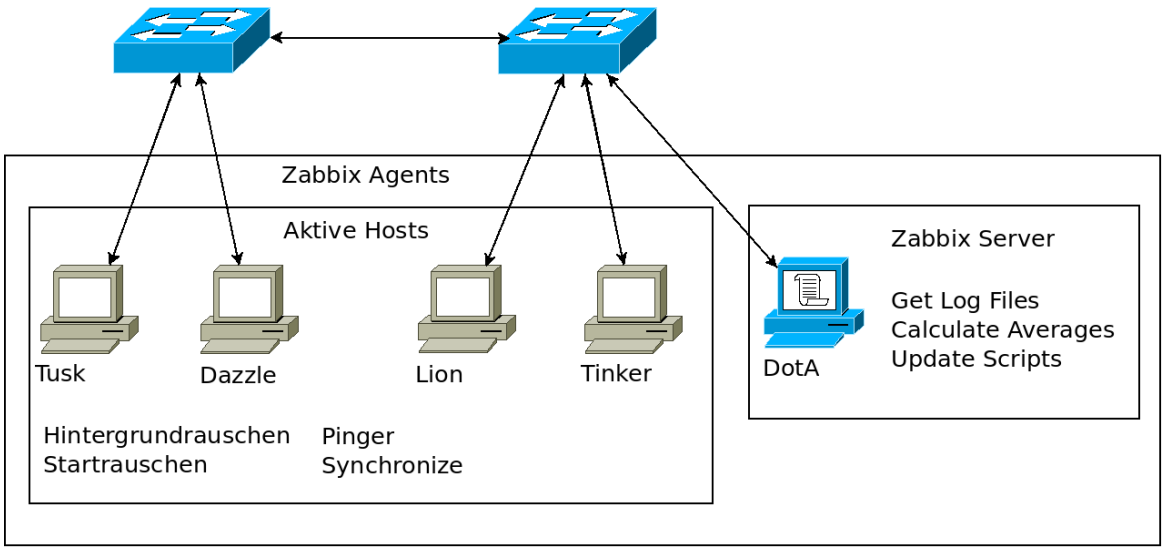

Abbildung 2.1: Aufbau des Netzwerks

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 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 4.1.1, Abschnitt 4.1.2 und Abschnitt 4.1.3 werden in Kapitel 5 miteinander verglichen.

4.1 Raspberry Pi Versuche

4.1.1 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 Abb. 7.1 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

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 4.1: Auslastung des Ethernet Ports bei 20 MB Paketen 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 4.2: Standardabweichung des der Last auf dem Ethernet Port bei 20 MB großen Paketen

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 2,0825% belastet ist.

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 was deren Last angeht. Die Standardabweichung σ der Agents bestätigt diese Annahme diese liegt nach Tabelle 4.2 bei 0,15411 Megabit.

In der Abb. 7.2 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 [**san:sd**] liegen.

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 4.3: 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 4.4: CPU Last Verteilung

4.1.2 200 Megabyte Testlauf

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.

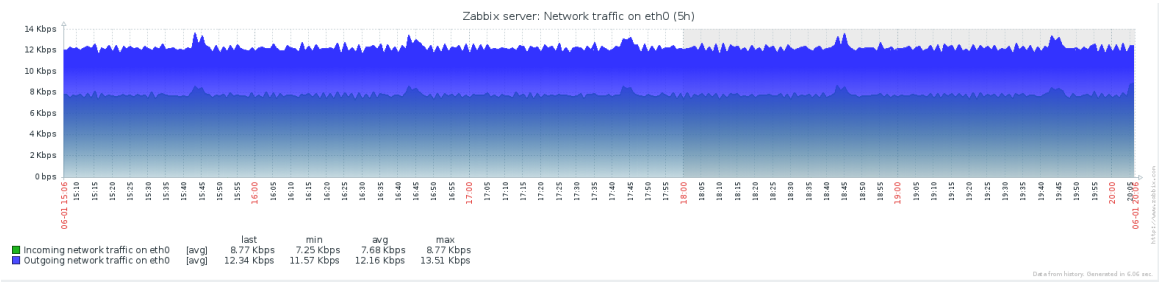

Abbildung 4.1: Traffic auf Eth0 beim Zabbix Server bei keinem Datenaustausch

Die Tabelle 4.6 zeigt die durchschnittlichen Übertragungswerte der einzelnen Pis auf, wie 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.

4.1.3 2 Gigabyte Testlauf

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. 4.2 nahelegt.

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 4.5: 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 4.6: 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 4.7: Standarbweichung bei keinem Datenaustausch

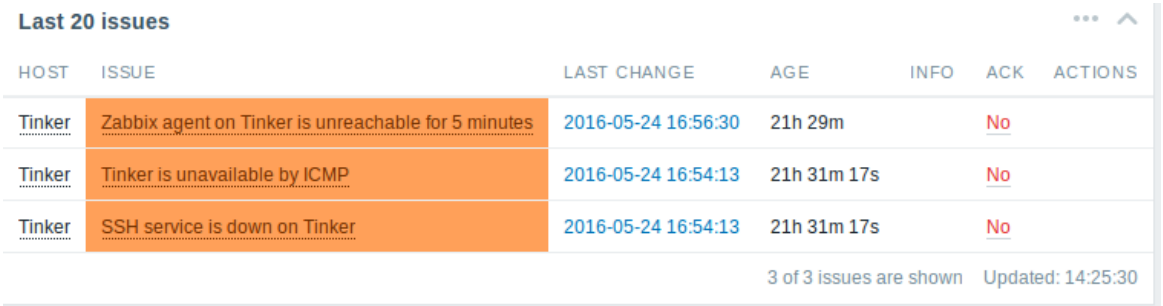

Abbildung 4.2: Fehlermeldung auf dem Zabbix Dashboard bezüglich Tinker

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 4.8: 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 4.9: Doppelte IP Standarbweichung der Werte

Wie man der Tabelle 4.8 sehen kann wird Dazzle mehr belastet als sonst. Vergleicht man die Standarbweichung von Tabelle 4.9 mit der Tabelle 4.2 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.

Kapitel 5

Vergleich VM/HW

5.1 Versuchsergebnisse

5.2 Kosten nutzen Faktor

Kapitel 6

Fazit

Kapitel 7

Anhang

7.1 20 Megabyte Test

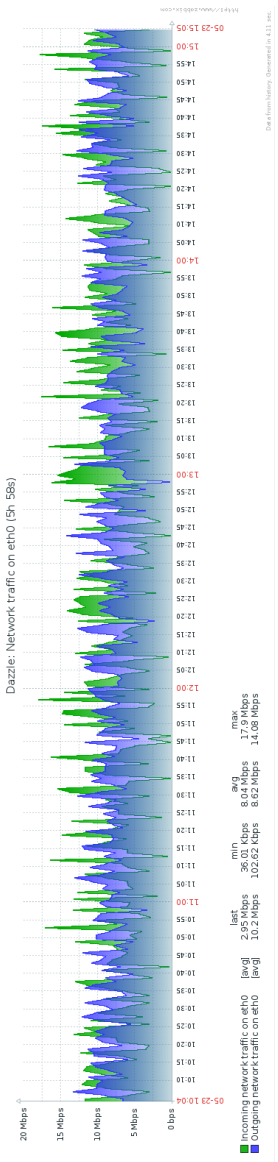

Abbildung 7.1: Traffic auf Eth0 bei Normalbetrieb auf Dazzle

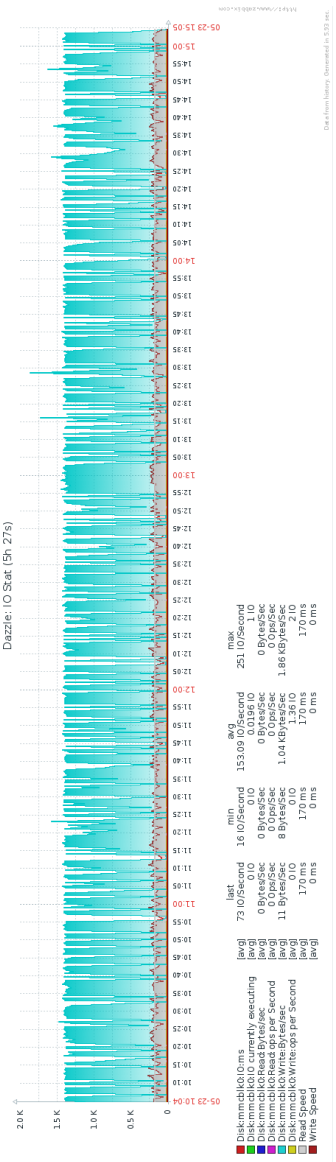

Abbildung 7.2: I/O Stats von Dazzle beim Normalbetrieb

Literatur

[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/.