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 Abschnitt 3.1 beschriebenen Versuche wurden mit folgender Hardware durchgeführt.

- 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.

2.2 Aufbau der Software

Das Testframework besteht aus drei verschiedenen Teilen.

- 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 [SIA16]. 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. Über die API Schnittstelle ist es möglich Programme für den Server zu schreiben und diese auszuführen. Der Zabbix Server ist ein sich selbst überwachender Agent. Dieser ist jedoch nicht beteiligt im allgemeinen Prozess des Frameworks.

Zabbix Agent Die Clients, die im Netzwerk überwacht werden sollen sind sogenannten Agents, die an den Server die Informationen weiterleiten, die vom Server gefordert werden.

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 ist 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 und dessen Größe.

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 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.

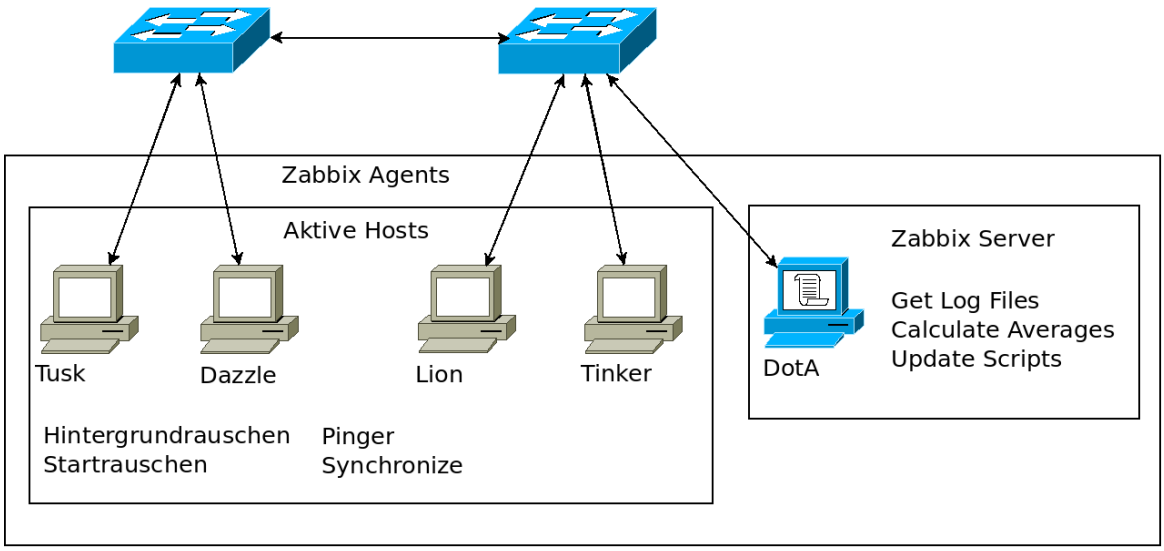

Abbildung 2.1: Aufbau des Netzwerks

Kapitel 3

Testaufbau

Die beste Art ein Netzwerktestframework zu testen, ist in dem man das Framework benutzt. Dazu wurden am Anfang der Arbeiten mehrere Tests definiert. Jeder dieser Tests ist eine in einem Netzwerk mögliche Fehlerquelle, die die Leistung des Netzwerkes beeinträchtigen kann. Um Fehler identifizieren zu können müssen jedoch auch die Normalwerte bestimmt werden. Dazu muss zunächst ein störungsfreies Netzwerk aufgebaut werden und dieses muss beobachtet werden. Die in diesem Netz gesammelten Werte stellen die Basis für die späteren Ergebnisse dar. Im zweiten Schritt wird das Netz mit Fehlern aufgebaut und zu beobachtet. Die somit gewonnen Werte eines fehlerbehafteten Netzwerkes können nun mit dem fehlerfreien Netzwerk vergleicht werden.

3.1 Testbeschreibung

In diesem Abschnitt sind die Tests beschrieben die im Rahmen der Arbeit durchgeführt worden sind. Alle Tests sind für eine Dauer von fünf Stunden ausgelegt.

Normalbetrieb: Dieser Test erzeugt die Werte mit denen die Fehlerbehafteten Tests verglichen werden. Dazu wird das Netzwerk wie in Abb. 2.1 verwendet.

Ethernetkabel ohne Isolierung: Ethernetkabel werden häufig sehr strapaziert;dies kann dazu führen, dass sich die Isolation löst und somit Strahlung einwirken kann. Dies kann die Übertragungsqualität im Netzwerk beeinträchtigen.

Falsch gedrehtes Twisted Pair Kabel: Twisted Pair Kabel gehören zu den gängigsten Kabeln des Ethernet Standards [Tan03], welche durch eine Verdrillung der einzelnen Kabel einen erhöhten Einstrahlungsschutz bilden.

Loop: Ein Loop ist ein mit sich selbst verbundener Switch. Der Switch beginnt dann über eine Broadcast Adresse Pakete zu verschicken, was beim Adress Resolution Protocol passiert. So wird über Anschluss A an Anschluss B ein Paket verschickt, dies löst bei Anschluss B

eine Reaktion aus. Anschluss B Antwortet auf die Anfrage vom Anschluss A. Dadurch entsteht eine Schleife aus Paketen, die das Netzwerk. blockiert. und so der Switch sich selbst blockiert und das Netzwerk komplett ausfällt. In dem in Abb. 2.1 gezeigten Versuchsauf-

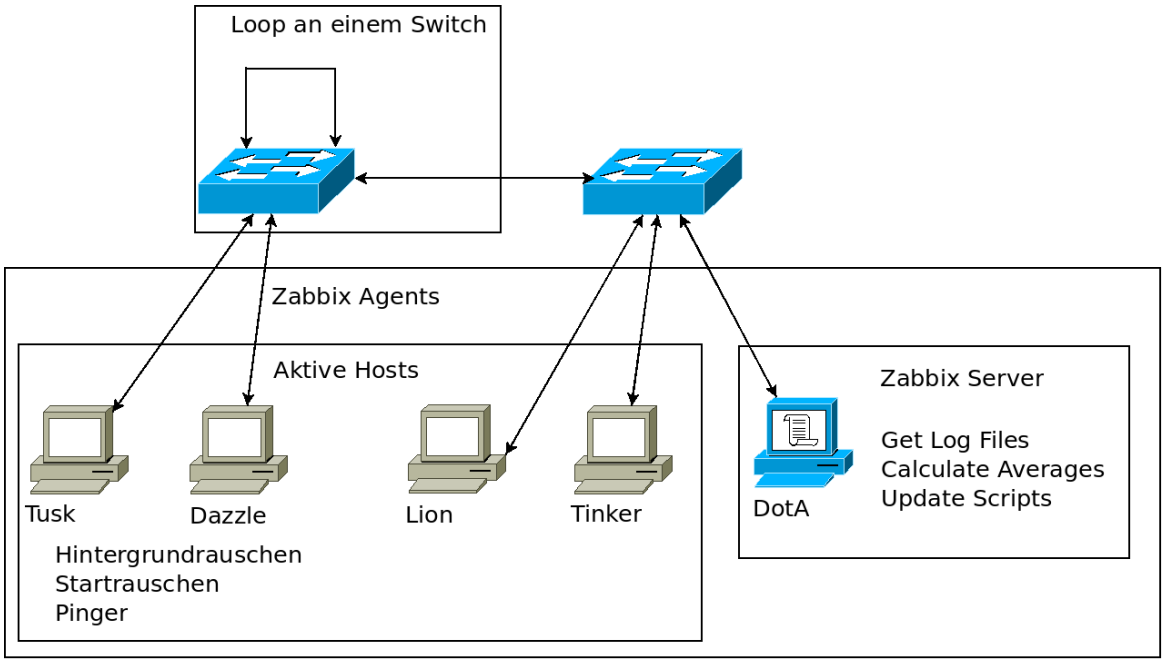

Abbildung 3.1: Ein Loop mit sich selber

bau ist jedoch noch eine weitere Form des Loops mögliches, es ist möglich beide Switches miteinander über zwei Ethernetkabel zu verbinden. Dieser Versuch hat den Aufbau wie in Abb. 3.2

Nicht angeschlossenes Kabel: Wie verhält sich das Netzwerk wenn ein Kabel von einem Endgerät im laufenden Betrieb entkoppelt wird.

Forkbomb: Unter einer Forkbomb versteht man ein Programm, das von sich selbst rekursiv Kopien erstellt so dass der Computer all seine Resourcen dazu verwendet weitere Kindprozesse von sich selber zu erzeugen.

:(){ :|:&};: [Wik15]

Festplatte: In einem Netzwerk werden ständig Daten versandt, was also passiert wenn die Festplatte von einem Rechner volläuft und der PC somit Arbeitsunfähig wird.

IP Adresse doppelt belegt: In einem Netzwerk besitzen alle Geräte eine im Idealfall, einmalige IP Adresse über diese Geräte ansrechbar sind. In privaten Haushalten, wird die vergabe von IP Adressen über das DHC Protokoll [Dro97]

Kollisionsdomänen: Kollisionsdomänen sind bereiche in einem Netzwerk wo sich mehrere Rechner eine eine Leitung teilen, dies ist geschieht in der Physikalischen Ebene eines Netwerks. Die Frage ist kann man eine Kollisionsdomäne erkennen.

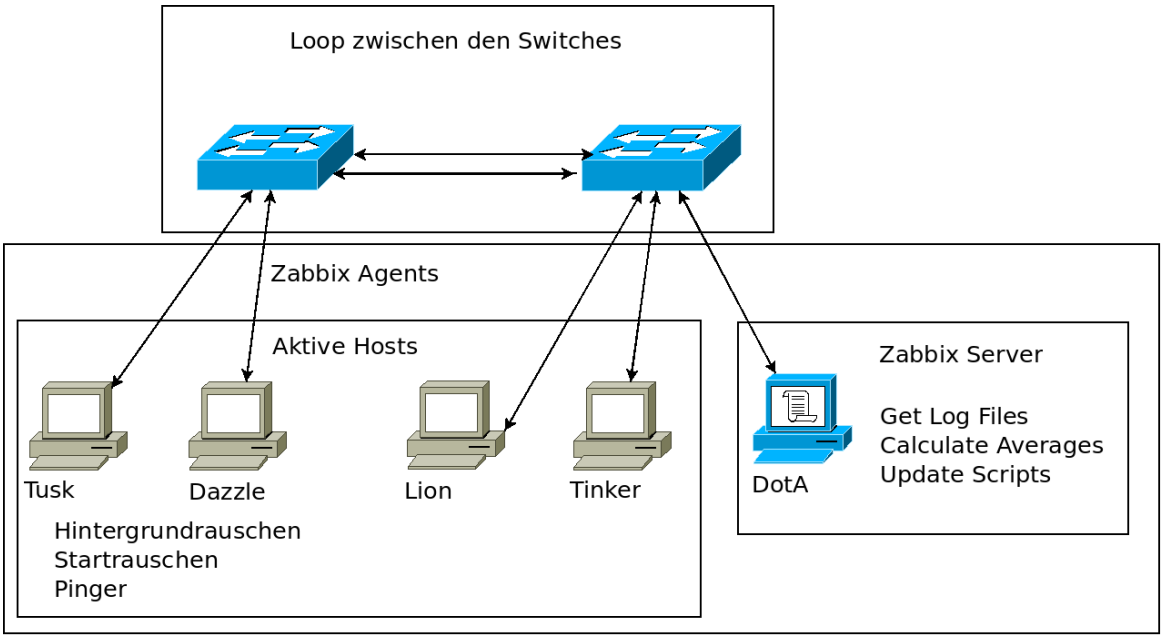

Abbildung 3.2: Ein Loop zwischen beiden Switches

3.2 Pis Real

3.3 Pis Virtuell

Kapitel 4

Versuche

4.1 Raspberry Pi Versuche

4.1.1 Durchgeführte Tests

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.

4.1.2 Normalbetrieb

Der erste durchgeführte Test ist der Normalbetrieb. In diesem Test ist das Ziel zu sehen wie sich das Netzwerk verhält wenn keine Störungen stattfinden. Das ermöglicht es das aussagen über das Netzwerk zu machen 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 erhalten wir einen konstant gleichbleibend großen Datenstrom. Als erste Kontroll Instanz werden die Daten die vom Zabbix erzeugt werden. Dazu werden die von Zabbix erzeugten Graphen rangezogen. Es werden immer die Graphen sein die den Traffic auf dem Netzwerk Port Eth0 und die Last des Schreibens auf der Festplatte des Pis sein.

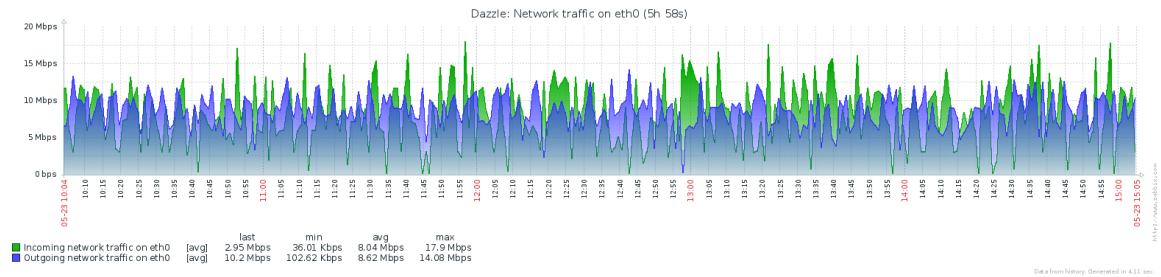

Abbildung 4.1: Traffic auf Eth0 beim Pi Dazzle

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

[Dro97] R. Droms. *Dynamic Host Configuration Protocol*. [Online; Stand 24. Mai 2016]. 1997. URL: `https://tools.ietf.org/html/rfc2131`.

[LLC16] Nagios Enterprises LLC. *Nagios*. [Online; Stand 11. Mai 2016]. 2016. URL: `https://de.wikipedia.org/wiki/Nagios`.

[SIA16] Zabbix SIA. *Zabbix*. [Online; Stand 11. Mai 2016]. 2016. URL: `http://www.zabbix.com/`.

[Tan03] Andrew S. Tanenbaum. *Computernetzwerke*. PEARSON Studium, 2003.

[Wik15] Wikipedia. *Forkbomb — Wikipedia, Die freie Enzyklopadie*. [Online; Stand 11. Mai 2015]. 2015. URL: `https://de.wikipedia.org/wiki/Forkbomb`.