

Automated Troubleshooting



Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2018

Autoren:	Raphael Jöhl und Nico Vinzens
Betreuer:	Prof. Beat Stettler
Experte:	Rolf Schaerer
Gegenleser:	Prof. Dr. Farhad D. Mehta

Aufgabenstellung BA

In einem Data-Center oder Campus-Netz passieren Fehler, welche oft erst spät erkannt werden, so dass sie negative Auswirkungen auf das Business haben, wie zB. dass sich ein Kunde beschwert, oder dass Geschäftsapplikationen nicht mehr funktionieren. Das Ziel der vorausgehenden Studienarbeit war, solche Störungen schnellstmöglich zu entdecken und automatisch zu beheben. Daraus ist OATS (**O**pen **S**ource **A**utomated **T**roubleshooting **S**ystem) entstanden, ein verteiltes System mit Sensoren, die Events auslösen, welche ihrerseits Troubleshooting Workflows auslösen, die versuchen das Problem automatisch zu beheben oder einen Techniker zu alarmieren.

In der nachfolgenden Bachelorarbeit soll OATS erweitert werden, damit eine grössere Bandbreite von Fehlern erkannt und automatisch behoben werden kann.

Konkret sollen:

- weitere Sensoren analysiert und ins System eingebunden werden
- die Events verschiedener Sensoren sollen intelligent korreliert werden können
- Troubleshooting Workflows sollen so erweitert werden, dass sie die Events der neuen Sensoren mitberücksichtigen
- Ein grafisches User-Interface soll den Zustand des Systems anzeigen

Um die Funktionstauglichkeit des erweiterten OATS Systems zu prüfen, sollen entsprechende Troubleshooting Use-Cases im Labornetzwerk ausgelöst und von OATS behandelt werden.

Rapperswil, 20.2.2018

Name: STETTLER BEAT



Unterschrift



Automated Troubleshooting

Abstract

Das Troubleshooting von Problemen in Netzwerken ist heutzutage zum grössten Teil immer noch manuelle Arbeit. Der Netzwerk Engineer sammelt dabei Informationen, indem er den Zustand der Verbindungen überprüft oder die Konfiguration der Geräte durchforstet.

Dieses altmodische Paradigma soll durch ein System ersetzt werden, welches das Troubleshooting automatisieren kann. Dafür wurde in der vorangegangenen Studienarbeit OATS (**O**pen **S**ource **A**utomated **T**roubleshooting **S**ystem) entwickelt, welches bereits in der Lage ist, eine gewisse Art von Fehler zu erkennen und zu behandeln. Ziel dieser Bachelorarbeit war es, dieses System zu erweitern und die Handhabung für die möglichen Benutzer zu vereinfachen.

Zu diesem Zweck wurde OATS um zwei Netzwerk-Sensoren erweitert: Streaming-Telemetry und Netflow, welche den bisherigen Sensor Syslog ergänzen. Für Streaming-Telemetry wurde ein eigener Client geschrieben, welcher in der Lage ist die benötigten Verbindungen zu den Netzwerkgeräten aufzubauen. Für Netflow wurde eine bereits existierende Open Source Lösung in OATS integriert. Da die neuen Sensoren in einem grossen Netzwerk theoretisch sehr grosse Datenmengen liefern, ist es essentiell, dass OATS mit diesem Big Data Problem umgehen kann. Deswegen wurde das bestehende Datenbankkonzept überarbeitet, indem neu InfluxDB und Kafka in das System eingegliedert wurden. So ist OATS dank Kafka in der Lage, die Daten verteilt zu verarbeiten und zu einem späteren Zeitpunkt zentral in InfluxDB persistent zu speichern. In OATS wird dieser sogenannte «Data Lake» zur Visualisierung der Daten benutzt.

Durch die Kombination von verschiedenen Open Source Tools und Technologien wurde somit ein System entwickelt, mit welchem es möglich ist, Probleme in Netzwerken automatisch zu erkennen und zu behandeln. Um aufzuzeigen, was mit diesen beiden Sensoren neu möglich ist, wurde ein Troubleshooting-Workflow umgesetzt, welcher sich den Streaming-Telemetry und Netflow-Daten bedient. Weiter wurde mit Grafana eine graphische Oberfläche zur Anzeige der erhaltenen Daten hinzugefügt und das Erstellen von neuen Troubleshooting Workflows wurde vereinfacht.

Da die Bachelorarbeit in einem Lab-Netzwerk entwickelt wurde, wäre der nächste logische Schritt nun, OATS einem Test in einem realen Umfeld zu unterziehen. Dort könnte man zum Beispiel die Daten einem Machine-Learning Algorithmus zuführen, um dynamisch Rückschlüsse über den Zustand des Netzes treffen zu können.

Automated Troubleshooting

OATS

Open Source **A**utomated **T**roubleshooting **S**ystem

Diplomanden	Nico Vinzens, Raphael Jöhl
Examinator	Beat Stettler
Experte	Rolf Schaerer
Themengebiet	Networks, Security & Cloud Infrastructure

Ausgangslage

In einem Data-Center oder Campus-Netz passieren Fehler, welche oft erst spät erkannt werden, so dass sie negative Auswirkungen auf das Business haben, wie z.B., dass sich ein Kunde beschwert, oder dass Geschäftsapplikationen nicht mehr funktionieren. Das Ziel der vorausgehenden Studienarbeit war, solche Störungen schnellstmöglich zu entdecken und automatisch zu beheben. Daraus ist OATS (**O**pen **S**ource **A**utomated **T**roubleshooting **S**ystem) entstanden, ein verteiltes System mit Sensoren, die Events auslösen, welche ihrerseits Troubleshooting Workflows auslösen, die versuchen das Problem automatisch zu beheben oder einen Techniker zu alarmieren.

In der nachfolgenden Bachelorarbeit soll OATS erweitert werden, damit eine grössere Bandbreite von Fehlern erkannt und automatisch behoben werden kann.

Konkret sollen:

- weitere Sensoren analysiert und ins System eingebunden werden
- die Events verschiedener Sensoren sollen intelligent korreliert werden können
- Troubleshooting Workflows sollen so erweitert werden, dass sie die Events der neuen Sensoren mitberücksichtigen
- Ein grafisches User-Interface soll den Zustand des Systems anzeigen

Um die Funktionstauglichkeit des erweiterten OATS Systems zu prüfen, sollen entsprechende Troubleshooting Use-Cases im Labornetzwerk ausgelöst und von OATS behandelt werden.

VORGEHEN/TECHNOLOGIEN

In einer ersten Analysephase wurden diverse potenzielle neue Sensoren darauf analysiert welchen zusätzlichen Nutzen diese bringen und inwiefern diese in das System integriert werden können.

Auf diese Analyse stützend wurden folgende Sensoren ausgewählt:

- **Streaming-Telemetry:** Ermöglicht das kontinuierliche Sammeln von Status-Daten eines Netzwerkgeräts.
- **Netflow:** Liefert Informationen zum aktuellen Netzwerkverkehr.

Diese zwei neuen Sensoren ergänzen den bisherigen Sensor Syslog, indem dass sie eine komplett neue Art von Informationen liefern, nämlich kontinuierliche Daten an Stelle von einzelnen Meldungen. Allerdings müssen diese Daten noch evaluiert werden, um mögliche Events zu erkennen.

Um mit diesen Daten umgehen zu können, wurden noch weitere Tools in OATS integriert:

- **Kafka:** Ermöglicht das verteilte Verarbeiten der Daten in «Pipelines», sodass alle Daten von unterschiedlichen Quellen an der richtigen Stelle landen.
- **Kafka-Streams:** Ermöglicht das kontinuierliche Evaluieren der Daten in Echtzeit.
- **InfluxDB:** Datenbank welche darauf spezialisiert ist, mit zeit-basierten Daten umzugehen.
- **Netbox:** Tool zum Verwalten eines Netzwerkinventars, fungiert als «Source of Truth» für den Netzwerkzustand in OATS.
- **Grafana:** erlaubt es uns im Zusammenspiel mit InfluxDB die Daten in Echtzeit in Dashboards zu visualisieren, sodass jederzeit der Zustand des Netzwerks einsehbar ist. Das Dashboard, welches für OATS entwickelt, ist in einer Abbildung weiter unten einsehbar.

Diese neuen Technologien komplettieren OATS zusammen mit denjenigen, die bereits während der vorangegangenen Studienarbeit integriert wurden:

- **Syslog:** Löst Events direkt auf den Netzwerkgeräten aus, z.B. wenn ein Netzwerkinterface runterfährt.
- **Salt:** Entscheidet aufgrund der erhaltenen Daten, welcher Workflow ausgeführt werden muss und führt diese aus. Innerhalb dieser Workflows werden Informationen gesammelt und Befehle an die Netzwerkgeräte gesendet.

Ergebnisse

Durch das Hinzufügen der beiden neuen Sensoren ist OATS in der Lage, eine grössere Bandbreite an möglichen Fehlern zu entdecken. Dies wurde mit einem exemplarischen Workflow, welcher Streaming-Telemetry- und Netflow-Daten korreliert, aufgezeigt.

Beim Entwickeln der Software wurde speziell darauf geachtet, dass die Umsetzung solcher neuer Workflows möglichst einfach gelingt. Dies wurde dadurch sichergestellt, dass OATS durch eine Konfigurationsdatei an das jeweilige Netzwerkumfeld angepasst werden kann. Das Einzige was zum Ende der Bachelorarbeit noch Programmierkenntnisse voraussetzt, ist das Umsetzen komplexer Troubleshooting-Workflows.

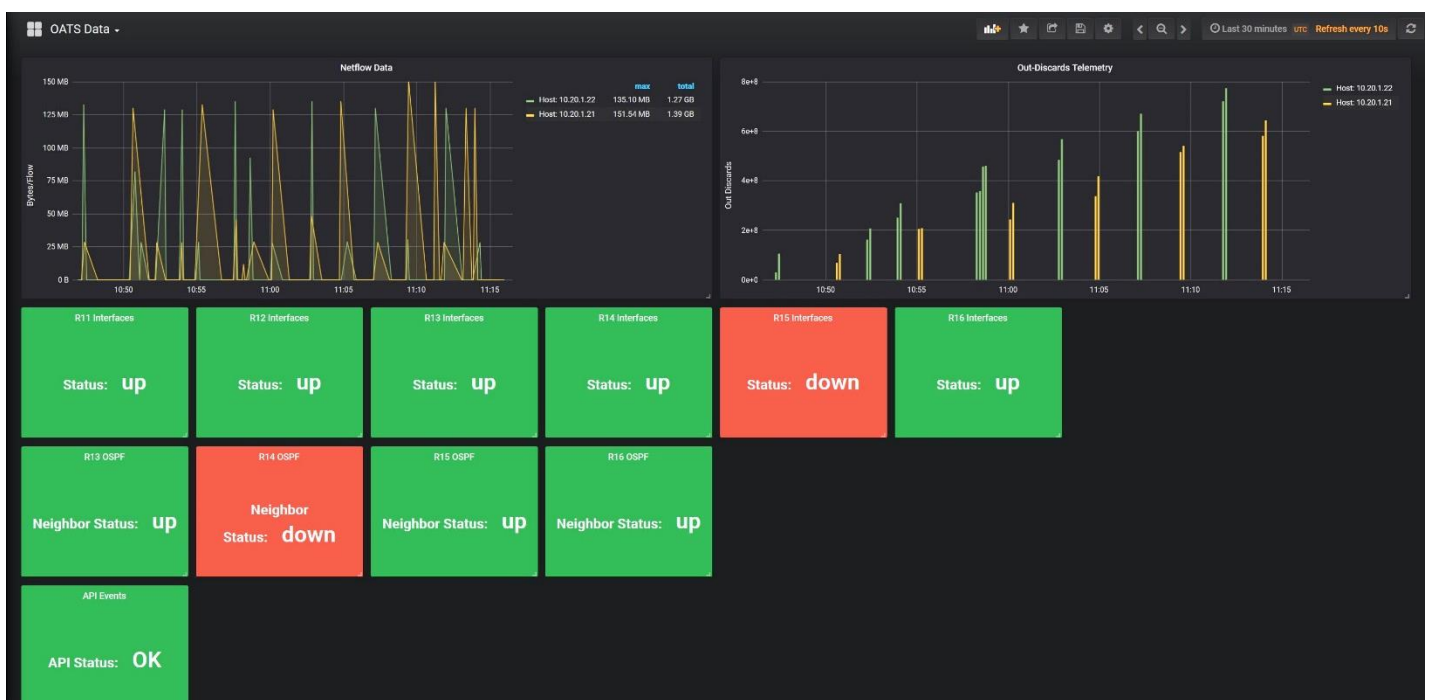
Im Verlauf der Arbeit hat sich herausgestellt, dass eines der Tools, Salt, an seine Grenzen stösst, wenn das Troubleshooting zeitkritisch ist. Dieses Tool müsste bei einer allfälligen weiterführenden Arbeit neu evaluiert und eventuell ersetzt werden.

Ausblick

Das Projektteam zeigt sich mit OATS zufrieden und ist der Meinung es wäre bereit in einem echten Netzwerk getestet zu werden.

Dort könnten unter anderem noch folgende Funktionalitäten hinzugefügt werden:

- **Machine-Learning:** Machine-Learning setzt reale Daten voraus, welche während der Entwicklung von OATS nicht verfügbar waren. Diese sind in einem realen Umfeld vorhanden und deswegen würde sich das integrieren von Machine-Learning lohnen.
- **Interaktives Troubleshooting:** Im Moment wird das Troubleshooting durchgeführt ohne dass manuell eingegriffen werden kann. Manchmal wäre es aber durchaus sinnvoll, dass sich OATS von einem Menschen bestätigen lässt, ob eine Änderung sinnvoll ist (→ Chat-Ops).
- **Fehlerklassifizierung:** Zurzeit wird in OATS angenommen, dass wenn ein Fehler eintritt, dass dieser schwer genug wiegt, dass er mittels automatischem Troubleshooting behandelt werden muss. In der Realität gibt es aber eine grosse Bandbreite an Fehlern, welche alle unterschiedlich dringend sind. Die Grundlage dafür wurde in OATS schon gesetzt, muss aber noch ausgearbeitet werden.



Grafana Dashboard: Zu diesem Zeitpunkt ist auf einem Router ein Problem vorhanden.



Inhaltsverzeichnis

I. Technischer Bericht	7
1. Ausgangslage & Problembeschreibung	8
2. Lösungskonzept	8
2.1 Sensoren.....	8
2.1.1 Streaming-Telemetry.....	8
2.1.2 Netflow	9
2.2 Event-Korrelation	9
2.2.1 Regel-basierte Korrelation.....	10
2.2.2 Machine-Learning.....	10
2.3 Troubleshooting-Workflows	10
2.4 Datenhaltung	10
2.5 Lab-Netzwerk	11
3. Umsetzung.....	12
3.1 Kafka.....	12
3.2 Syslog.....	13
3.3 Napalm-logs	13
3.4 Netconf: Streaming-Telemetry	13
3.5 vflow.....	13
3.6 Kafka Streams.....	13
3.7 OATS-API	14
3.8 Event-Korrelation	14
3.9 Salt.....	14
3.10 Napalm	14
3.11 InfluxDB.....	14
3.12 Netbox.....	14
3.13 Grafana.....	15
4. Ergebnis.....	15
5. Ausblick	16
6. Danksagung	16
II. Software-Engineering Dokumente.....	17
1. Domainanalyse	18
1.1 Einführung.....	19



1.1.1 Zweck.....	19
1.1.2 Gültigkeitsbereich.....	19
1.2 Technologieanalyse.....	20
1.2.1 Ping.....	20
1.2.2 Traceroute	23
1.2.3 SNMP – Simple Network Management Protocol	25
1.2.4 NETCONF – Network Configuration Protocol.....	27
1.2.5 YANG.....	28
1.2.6 Syslog.....	31
1.2.7 Netflow	33
1.2.8 Netzwerk-Sniffing	35
1.2.9 IP SLA - Internet protocol service level agreement	35
1.2.10 SSH – Secure Shell	37
1.2.11 Streaming-Telemetry.....	37
1.3 Problemanalyse.....	41
1.3.1 OSI-Modell.....	41
1.3.2 Probleme	42
1.4 Deviceanalyse.....	50
1.4.1 Geräte in einem Netzwerk	50
1.4.2 Probleme durch Geräte	50
1.5 Erweiterungsanalyse	51
1.5.1 Event-Korrelation	51
1.5.2 Datenbank	55
1.5.3 Codebasis.....	61
1.5.4 Sensoren.....	62
1.5.5 Deployment Automatisierung.....	64
1.5.6 User Interface	65
1.5.7 Weitere Troubleshooting-Fälle.....	66
1.5.8 Weiterführende Themen.....	70
2. Anforderungen	71
2.1 Zweck	72
2.1.1 Beschreibung	72
2.1.2 Gültigkeitsbereich.....	72
2.1.3 Übersicht	72
2.2 Systemumfang	73
2.2.1 Stakeholder	73
2.2.2 Systemfunktionalität	73



2.2.3 Systemumfeld.....	73
2.2.4 Architekturbeschreibungen.....	74
2.2.5 Einschränkungen	76
2.2.6 Annahmen	76
2.2.7 Abhängigkeiten.....	77
2.3 Use Cases	78
2.3.1 Aktoren.....	78
2.3.2 Use Case Diagramm.....	78
2.3.3 Use Case Diagramm nach Kategorien	79
2.3.4 Use Cases im Brief Format.....	79
2.3.5 Zusätzliche Use Cases in der BA	82
2.4 Workflow Anforderungen.....	84
2.4.1 Abhängigkeiten.....	84
2.4.2 BGP	84
2.4.3 Wireless	84
2.4.4 Campus QoS	84
2.4.5 Datacenter	84
2.4.6 Dauerhafte Netzwerkverkehrsspiegelung.....	85
2.4.7 OSPF.....	85
2.4.8 Spanning-Tree.....	85
2.4.9 Event-Korrelation	85
2.5 Nicht funktionale Anforderungen.....	86
2.5.1 Modifizierbarkeit.....	86
2.5.2 Interoperabilität	86
2.5.3 Richtigkeit.....	86
2.5.4 Skalierbarkeit.....	86
3. Architektur	87
3.1 Einführung.....	88
3.1.1 Zweck.....	88
3.1.2 Gültigkeitsbereich.....	88
3.2 Toolanalyse	89
3.2.1 Sensoren.....	89
3.2.2 NAPALM.....	91
3.2.3 NAPALM-Logs	93
3.2.4 Apache Kafka	95
3.2.5 Datenanalyse	96
3.2.6 vflow	101



3.2.7 Troubleshooting- und Konfigurationsmanagement-Tools	101
3.2.8 ZeroMQ.....	111
3.2.9 Slack.....	111
3.2.10 Datenspeicherung	112
3.2.11 Grafana	113
3.2.12 Flask	113
3.3 Architektur: Stand Studienarbeit / Beginn Bachelorarbeit.....	114
3.3.1 Erfassung / Sensoren	115
3.3.2 Identifizierung	115
3.3.3 Klassifizierung	115
3.3.4 Behebung.....	115
3.3.5 Überprüfung	115
3.3.6 Reporting/Logging	115
3.4 Architektur: Bachelorarbeit	116
3.4.1 Netconf / Streaming Telemetry.....	117
3.4.2 Netflow / vflow	117
3.4.3 Syslog / Napalm-logs	117
3.4.4 OATS-API.....	117
3.4.5 Kafka	117
3.4.6 Kafka-Streams.....	117
3.4.7 Event-Korrelation	118
3.4.8 InfluxDB	118
3.4.9 Netbox	118
3.4.10 Grafana	118
3.4.11 Napalm & Salt.....	118
3.5 Abläufe	119
3.5.1 Syslog basierter Workflow.....	119
3.5.2 Streaming-Telemetry basierter Workflow	120
3.6 Logische Architektur	121
3.6.1 napalm-logs	122
3.6.2 oats_api	122
3.6.3 telemetry	122
3.6.4 streaming_clients	123
3.6.5 kafka	124
3.6.6 salt	124
3.6.7 oatsnetbox.....	125
3.6.8 oatspsql	125



3.6.9 oatsinflux	125
3.7 Schnittstellen	126
3.7.1 OATS <> User	126
3.7.2 OATS <> Sensoren	126
3.7.3 OATS <> Netzwerkgeräte	126
3.7.4 Interne Schnittstellen	127
3.8 Deployment.....	128
3.8.1 Server.....	128
3.8.2 Netzwerkdevice	128
3.9 Datenspeicherung	129
3.9.1 Device	129
3.9.2 Events	130
3.9.3 Cases.....	132
3.10 Prozesse und Threads	134
3.10.1 Prozesse.....	134
3.10.2 Threads.....	135
3.11 Grössen und Leistung.....	135
3.11.1 Salt-Troubleshooting-Workflows	135
3.11.2 Laden von Konfigurationen	136
3.11.3 Fazit	136
4. Anleitungen	137
4.1 Einführung.....	138
4.2 Installationsanleitung.....	139
4.2.1 Python	139
4.2.2 Pip.....	139
4.2.3 Apt Packages.....	139
4.2.4 Kafka	140
4.2.5 Influx.....	140
4.2.6 Postgres	140
4.2.7 Netbox	141
4.2.8 OATS	142
4.2.9 Vagrant	142
4.3 Konfigurationsanleitung.....	143
4.3.1 OATS	143
4.3.2 Napalm-Logs.....	146
4.3.3 Salt-Proxies	146
4.3.4 Netbox	147



4.3.5 Netzwerkgeräte	148
4.4 Benutzungsanleitung	150
4.4.1 OATS	150
4.4.2 Salt Environment	150
4.4.3 Netbox	151
4.4.4 PostgreSQL	151
4.4.5 Influx	152
4.4.6 API.....	152
4.4.7 Testing der Workflows	153
4.5 Development.....	154
4.5.1 Syslog.....	154
4.5.2 Napalm-Logs	154
4.5.3 OATS napalm logs Client.....	154
4.5.4 Kafka Consumer.....	154
4.5.5 Salt	157
4.5.6 Streaming-Telemetry.....	158
4.5.7 Systemd	159
4.5.8 Kafka-Streams.....	159
4.5.9 Netbox	159
III. Glossar	160
1. Begriffe	160
2. Abkürzungen	162
IV. Bibliographie.....	164
1. Quellenverzeichnis	164
2. Abbildungsverzeichnis	167
V. Anhänge.....	169
Anhang A: Projektplan.....	170
Anhang B: Technische Risiken	188



I. Technischer Bericht



Projekt: Automatisiertes Troubleshooting

Raphael Jöhl
Nico Vinzens



1. Ausgangslage & Problembeschreibung

In einem Data-Center oder Campus-Netz passieren Fehler, welche oft erst spät erkannt werden, so dass sie negative Auswirkungen auf das Business haben, wie z.B., dass sich ein Kunde beschwert, oder dass Geschäftsapplikationen nicht mehr funktionieren. Das Ziel der vorausgehenden Studienarbeit war, solche Störungen schnellstmöglich zu entdecken und automatisch zu beheben. Daraus ist OATS (**O**pen **S**ource **A**utomated **T**roubleshooting **S**ystem) entstanden, ein verteiltes System mit Sensoren, die Events auslösen, welche ihrerseits Troubleshooting-Workflows auslösen, die versuchen das Problem automatisch zu beheben oder einen Techniker zu alarmieren.

In der nachfolgenden Bachelorarbeit soll OATS erweitert werden, damit eine grössere Bandbreite von Fehlern erkannt und automatisch behoben werden kann.

Konkret sollen:

- weitere Sensoren analysiert und ins System eingebunden werden
- die Events verschiedener Sensoren sollen intelligent korreliert werden können
- Troubleshooting Workflows sollen so erweitert werden, dass sie die Events der neuen Sensoren mitberücksichtigen
- Ein grafisches User-Interface soll den Zustand des Systems anzeigen

Um die Funktionstauglichkeit des erweiterten OATS zu prüfen, sollen entsprechende Troubleshooting Use-Cases im Labornetzwerk ausgelöst und von OATS behandelt werden.

2. Lösungskonzept

2.1 Sensoren

Da mit Syslog [1] bereits ein Sensor in OATS integriert ist, welcher erkennt, wenn Events im Netz geschehen, wurde zuerst analysiert, was für Sensoren existieren, die uns andere Informationen über ein Netzwerk liefern können, als nur Events.

Das Problem bei Events ist, dass sie binär sind, d.h. wenn keine Events auftreten, werden keine Informationen über das Netz gesammelt. So könnte man auf den Trugschluss kommen, dass der Zustand des Netzes in Ordnung ist, solange keine Events von den Sensoren gemeldet werden.

Zu diesem Zweck wurden mögliche Sensoren in folgende zwei Kategorien aufgeteilt:

- **Zustandssensoren:** liefern Informationen über den Zustand des Netzes. Diese Informationen geben direkt an ob ein Event geschehen ist, oder nicht und müssen somit nicht noch zusätzlich ausgewertet werden. Idealerweise enthalten sie aber direkt Informationen zum Ursprung des Events. Syslog ist ein solcher Zustandssensor.
- **Daten-Sensoren:** liefern kontinuierlich Daten, welche ausgewertet werden müssen um Events im Netz zu erkennen. Solche Sensoren erlauben das Erkennen von «nicht-binären» Events, z.B. wenn die Last im Netz zu hoch wird.

Um Syslog zu ergänzen wurden somit mehrere mögliche Daten-Sensoren analysiert und am Ende wurde die Auswahl auf **Streaming-Telemetry** [2] und **Netflow** [3] eingegrenzt.

2.1.1 Streaming-Telemetry

Streaming-Telemetry ist ein neuer Ansatz zur Netzwerküberwachung, in welchem kontinuierlich Daten von Geräten in inkrementellen Updates gestreamt werden (Push anstatt Pull). Eine zentrale Sammelstelle kann dabei angeben, für welche Daten sie sich interessiert und erhält nur diese Daten. Die Daten sind als YANG-Modelle [4] modelliert und haben folgende Form:



```
"RootOper": {
  "InfraStatistics": {
    "GigabitEthernet0/0/0/0": {
      "Latest": {
        "GenericCounters": {
          "OutputQueueDrops": 0,
          "LastDiscontinuityTime": 1449091544,
          "InputIgnoredPackets": 0,
          "PacketsReceived": 40881,
          "OutputDrops": 0,
          "UnknownProtocolPacketsReceived": 0,
          "RuntPacketsReceived": 0,
          "CRCErrors": 0,
          "SecondsSinceLastClearCounters": 0,
          "CarrierTransitions": 0,
          "MulticastPacketsSent": 26784,
          "BytesSent": 43668872,
          "ThrottledPacketsReceived": 0,
          "Applique": 0,
          "FramingErrorsReceived": 0,
          "GiantPacketsReceived": 0,
          "OutputUnderruns": 0,
          "OutputErrors": 0,
          "BroadcastPacketsReceived": 0,
          "OutputBuffersSwappedOut": 0,
          "Resets": 0,
          "SecondsSincePacketSent": 0,
          "InputAborts": 0,
          "InputOverruns": 0,
          "InputQueueDrops": 0,
          "InputDrops": 0,
          "AvailabilityFlag": 0,
          "MulticastPacketsReceived": 26775,
          "SecondsSincePacketReceived": 0,
          "OutputBufferFailures": 0,
          "BytesReceived": 43635706,
          "ParityPacketsReceived": 0,
          "BroadcastPacketsSent": 2,
          "LastDataTime": 1449320856,
          "InputErrors": 0,
          "PacketsSent": 40904
        }...
      }
    }
  }
}
```

 IX Forum 9 © 2015 Cisco and/or its affiliates. All rights reserved. Cisco Confidential 12

Abbildung 1: Streaming-Telemetry Yang-Modell

Diese Daten eignen sich gut zum programmatischen Auswerten (anders als Syslog, welches plaintext-basiert ist). Hier sieht man auch direkt mögliche Daten, welche uns Aufschluss über den Zustand des Netzes geben können, z.B. «CRCErrors» oder «OutputQueueDrops».

2.1.2 Netflow

Netflow ist ein Feature welches die Möglichkeit bietet, Informationen über den Netzwerkverkehr zu sammeln, sobald Pakete ein Interface betreten und/oder verlassen. Die gesammelten Informationen werden in UDP Datagrammen exportiert. Netflow ist ein Messverfahren, d.h. es werden Informationen über den Verkehr gesammelt aber keine Änderungen vorgenommen. Router und Switches welche Netflow unterstützen, können auf allen Interfaces, auf denen Netflow aktiviert ist, IP Traffic Informationen sammeln.

Je nach Implementation des Herstellers, unterscheiden sich die Informationen, die in einem Network Flow enthalten sind. Unter Cisco beinhaltet ein Paket folgendes:

- IP Source Adresse (L3)
- IP Destination Adresse (L3)
- Source Port (L4)
- Destination Port (L4)
- Layer 3 Protocol
- Type/Class of Service
- Router/Switch Interface (L2)
- Timestamp: Start des Flows
- Timestamp: Ende des Flows

Pakete, die diese Informationen gemeinsam haben, können als Flow zusammengefasst werden. Speziell an diesen Flows ist, dass sie Informationen von den Netzwerk-Layer 2-4 beinhalten, was mit anderen Sensoren nicht möglich ist.

2.2 Event-Korrelation

Events können über Beziehungen untereinander verfügen [5] und somit ist es möglich, dass mehrere Problemmeldungen eingehen, die alle auf dasselbe Problem hindeuten. Eine solche Event-Korrelation, welche Schlüsse auf den Ursprung eines oder mehreren Events zieht, ist im Kontext eines automatischen Troubleshooting ein mächtiges Werkzeug.



Zu diesem Zweck wurden mögliche Arten der Event-Korrelation untersucht. Wobei die 2 vielversprechendsten nachfolgend erläutert werden.

2.2.1 Regel-basierte Korrelation

Dabei werden direkt im Code oder auch in einer externen Datei feste Abläufe festgelegt, was bei einem Event für Operationen und Aktionen anzuwenden sind. Ein solcher Regelsatz könnte so aussehen: «Wenn *Event A* 10-mal während 5 Minuten auftritt ist *Event B* eingetreten. Um das durch *Event B* eingetretene Problem zu lösen muss *Massnahme C* ausgeführt werden».

Für diesen Korrelations-Ansatz existiert eine Auswahl von verschiedenen Korrelations-Operationen:

- **Kompression:** Kompression mehrerer Events
- **Logische Operationen:** boolesche Logik (*AND*, *OR*, *EQUALS*, etc.)
- **Aggregation:** Aggregation mehrerer Events
- **Filtering:** Filtern von Events
- **Thresholding:** Erzeugen von Events, wenn ein Schwellenwert (*Threshold*) überschritten wird
- **Zeitliche Beziehungen:** es werden z.B. nur Events der letzten 20 Sekunden miteinander korreliert
- u.a.

2.2.2 Machine-Learning

Machine Learning bezeichnet das Konstruieren und Anwenden von Algorithmen, welche mittels Daten trainiert werden und versuchen Vorhersagen zu treffen. Um diese Vorhersagen zu treffen bedienen sich die Algorithmen der statistischen Mathematik und der Wahrscheinlichkeitstheorie. Leider sprengt dieses Thema den Rahmen dieser Bachelorarbeit, weswegen bei der Event-Korrelation nur die **Regel-basierte Korrelation** eingesetzt wurde.

2.3 Troubleshooting-Workflows

Ein Troubleshooting-Workflow beschreibt im Kontext dieser Arbeit das gesamtheitliche Bearbeiten eines Problemfalls im Netzwerk. So wurden zum Beginn der Arbeit einige mögliche Troubleshooting-Fälle definiert, welche durch OATS behandelt werden können sollen.

Da diese Fälle ziemlich theoretischer Natur sind, wurde entschieden, dass der Fokus nicht auf dem Bearbeiten möglichst vieler solcher Workflows liegen soll, sondern dass OATS so ausgelegt werden soll, dass möglichst einfach neue Troubleshooting-Workflows umgesetzt werden können.

2.4 Datenhaltung

Für das Visualisieren der Daten wurde nach einer Datenbanklösung gesucht, mit welcher möglichst einfach die bestehenden Datensätze innerhalb von OATS visualisiert werden können. So wurde als Datenbank für die Sensor-Daten InfluxDB [6] bestimmt. InfluxDB ist dafür ausgelegt, mit zeitbasierten Daten umzugehen, wie sie die Sensoren liefern. Jeder Datensatz in der InfluxDB verfügt dabei über folgende Felder:

- **sensor_type:** der Sensor, von welchem dieser Datensatz erfasst wurde (Syslog, Streaming-Telemetry oder Netflow).
- **event_name:** beschreibt das Event, welches dem Datensatz zugrunde liegt.
- **timestamp:** der Zeitpunkt an dem der Datensatz vom Sensor generiert wurde.
- **host:** Der Host, von welchem der Datensatz stammt.
- **severity:** die Schwere des Events im Datensatz.
- **data:** die Daten welche vom Sensor geliefert werden. Hat bei jedem Sensor-Typ ein unterschiedliches Format und muss deswegen auch unterschiedlich behandelt werden.

Dieses Format stellt sicher, dass neu in der Zukunft hinzugefügte Sensoren und deren Daten mit OATS kompatibel sind.



2.5 Lab-Netzwerk

Um OATS kontinuierlich zu testen, wurden von den Betreuern der Arbeit zwei Lab-Netzwerke aufgebaut.

Das eine stammt aus der Studienarbeit und wird ohne Änderungen so weiterverwendet, um sicherzustellen, dass alles was zum Ende der Studienarbeit funktioniert, auch noch am Ende der Bachelorarbeit funktioniert. Dieses Netzwerk ist rein virtuell.

Das zweite Netz wurde spezifisch dafür ausgelegt, einen Workflow zu entwickeln, der auf den Daten der neuen Sensoren basiert. Dieses Netz wurde mit physikalischen Geräten aufgesetzt, um realistischere Daten generieren zu können.

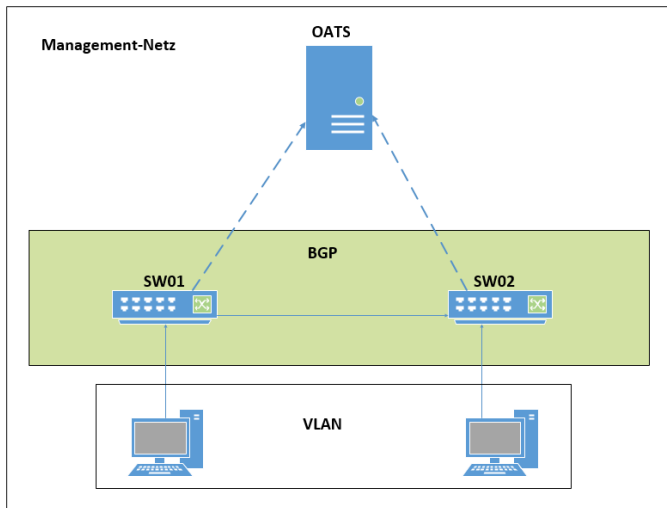


Abbildung 3: physikalisches Lab-Netz

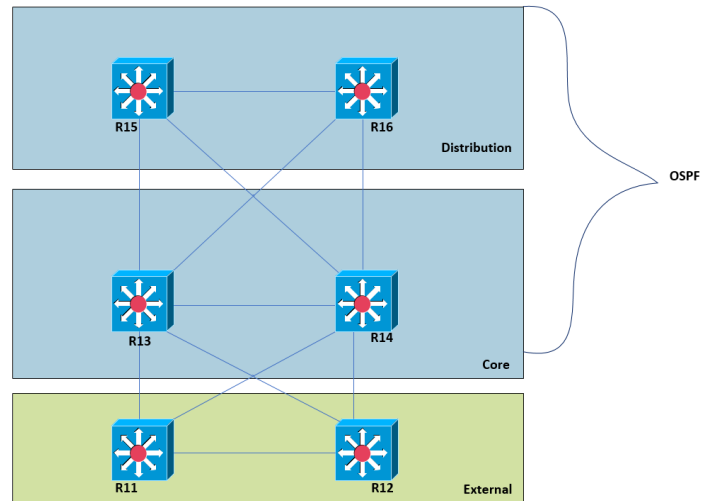


Abbildung 2: virtuelles Lab-Netz

3. Umsetzung

Mit diesem zugrundeliegenden Konzept wurden Tools identifiziert, welche diese Aufgaben erfüllen können und im weiteren Verlauf der Bachelorarbeit zu einem sinnvollen Ganzen zusammengeschlossen. Wo keine zufriedenstellenden Tools existieren, wurden diese selbst entwickelt:

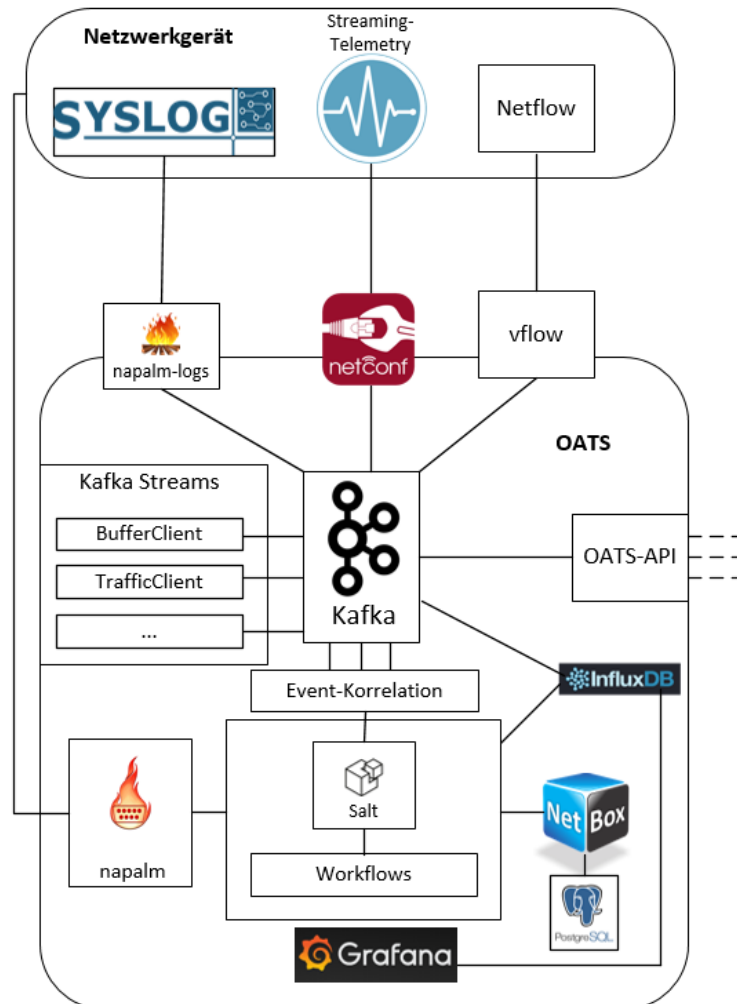


Abbildung 4: OATS-Architektur

3.1 Kafka

Apache Kafka [7] ist eine verteilte *Streaming-Plattform* und übernimmt innerhalb von OATS die Rolle einer Message-Queue und eines temporären Datenspeichers.

Die zwei wichtigsten Konzepte von Kafka sind *Topics* und *Records*:

- **Records:** Key/Value Dateninstanz welche über einen Timestamp verfügt. Enthalten die Daten/Events die von den Sensoren geliefert werden.
- **Topics:** Ein Topic ist ein Name für eine Kategorie oder einen *Feed* zu welchem ein Record veröffentlicht wird. Topics sind bei Kafka immer für mehrere Abonnenten ausgelegt, d.h. ein Topic kann über keinen, einen oder mehrere Konsumenten verfügen.

Kafka bildet die Lebenslinie von OATS ab, indem der grösste Teil der Kommunikation innerhalb des Systems über Kafka-Topics abgewickelt wird.



3.2 Syslog

Syslog ist ein Protokoll welches im Netzwerkumfeld dazu verwendet wird, Informationen zentral zu sammeln, der Syslog-Prozess läuft in der Regel direkt auf den Netzwerkgeräten.

Auf jedem Netzwerkgerät muss genau definiert werden, welche Meldungen geschickt werden und welche nicht. So kann z.B. eingestellt werden, dass wenn ein Interface auf einem Router runterfährt, automatisch eine Meldung von diesem Router an eine zentrale Stelle geschickt wird.

In diesem Projekt wurde das entwickelte OATS als zentrale Sammelstelle eingestellt. Somit schicken alle Netzwerkgeräte bei vordefinierten Events eine Meldung an OATS.

3.3 Napalm-logs

Da Syslog als Sensor im Netz eingesetzt wird, ist eine offene Frage, wie die plaintext-Nachrichten von Syslog in eine im Code nutzbare Form gebracht werden können. Diese Lücke füllt napalm-logs.

Napalm-logs [8] ist eine Open-Source Software, die als Daemon läuft und auf Syslog Meldungen horcht und diese in eine herstellerunabhängige Form bringt.

Die Meldungen werden entweder direkt durch TCP/UDP empfangen oder im Zusammenhang mit einem Messaging-System wie ZeroMQ [9] entgegengenommen und verschickt.

Für die Verwendung in OATS wurde zusätzlich ein kleiner Client geschrieben, welcher die Daten von napalm-logs entgegennimmt und in ein Kafka-Topic schreibt.

3.4 Netconf: Streaming-Telemetry

Für das Betriebssystem unserer Netzwerkgeräte existierte noch keine Lösung für Streaming-Telemetry, weswegen ein eigener *Telemetry-Client* entwickelt wurde.

Dabei teilt der Client dem Netzwerkgerät mit, welche Daten er in welchem regelmässigen Intervall er erhalten möchte.

Ein Benutzer von OATS kann dabei direkt in der Konfigurationsdatei angeben, welche Daten er von welchem Host abonnieren und in welchem Intervall er diese erhalten möchte. Das Beispiel in

```
hosts:
- host: 10.20.1.22
  port: 999
  username: user
  password: pw
- host: 10.20.1.21
  port: 1000
  username: user2
  password: pw2

subscriptions:
- subscription:
  xpath: /if:interfaces-state/interface/statistics/out-discards
  period: 1000
  kafka_publish_topic: interfaces-out-discards
  kafka_streams_eval: false
  correlate_event: false
```

Abbildung 5: minimale OATS-Konfigurationsdatei

Abbildung 5 zeigt eine minimale Version der Datei, in der von zwei Hosts, das gleiche Yang-Modell (liefert Daten zu den Interfaces des jeweiligen Hosts) abonniert. Wie man sieht, kann auch direkt angegeben werden ob die Daten evaluiert (*kafka_streams_eval*) und/oder korreliert (*correlate_event*) werden sollen. Für eine genauere Beschreibung, wie die Datei ausgelegt ist verweisen wir hierbei auf die Benutzungsanleitung.

3.5 vflow

Vflow [10] ist ein Open Source Netflow-Client, welcher in OATS integriert wurde. Er nimmt die Datensätze von Netflow entgegen und leitet sie an OATS weiter.

3.6 Kafka Streams

Kafka ermöglicht das Verarbeiten von Streaming-Daten, wie sie von Streaming-Telemetry geliefert werden, durch die Kafka Streams Library. Der Fokus liegt dabei auf einem einfachen Java-basierten Deployment.

Der entwickelte Kafka Streams Client entnimmt die Daten aus einem Kafka-Topic und vergleicht den aktuellsten Record mit vorherigen Record. Der Vergleich basiert auf Parametern, welche in der OATS-Konfigurationsdatei mitgegeben werden. Die Parameter enthalten einen Schwellenwert, einen



Vergleichsoperator (<, >, ==, usw.) und einen Kafka Topic-Namen in welches ein Event geschrieben wird.

3.7 OATS-API

Um das Hinzufügen von neuen Sensoren einfach zu gestalten, wurde eine API entwickelt, welche eine REST-Schnittstelle anbietet, die z.B. das manuelle Erstellen von Events in OATS erlaubt.

3.8 Event-Korrelation

Für die Event-Korrelation wurden 2 der oben beschriebenen Korrelations-Operationen entwickelt:

- **Aggregation:** es sind 2 Arten von Aggregation von Events möglich: Aggregation gleichartiger Events und Aggregation unterschiedlicher Events.
- **Kompression**

Es wurden nur diese 2 entwickelt, da diese in den vorhandenen Troubleshooting-Workflows benötigt werden.

3.9 Salt

Salt [11] ist eine komplexe Open Source Software (Saltstack ist die Enterprise Version) mit dem Ziel eine ereignisbasierte Netzwerkautomation zu erreichen. OATS bildet eine Art Wrapper um Salt und bedient sich der vielfältigen Funktionen zum Sammeln von Daten und Manipulieren von Netzwerkgeräten.

Salt (bzw. ein Salt-master) ist dafür verantwortlich als Reaktion auf ein von Syslog ausgelöstes und von napalm-logs verarbeitetes Event einen Troubleshooting Workflow auszuführen.

3.10 Napalm

Napalm [12] (Network Automation and Programmability Abstraction Layer with Multivendor support) ist eine Open-Source Python-Library, die Funktionen implementiert, welche es ermöglichen mit unterschiedlichen Netzwerkgeräten zu interagieren. So ermöglicht napalm den Verbindungsaufbau zu Geräten, die Manipulation deren Konfiguration und das Sammeln von Informationen.

Napalm ist direkt in Salt integriert, was die Ausführung der Funktionen einfach macht.

3.11 InfluxDB

Event Daten welche über Kafka von den Netzwerksensoren geliefert werden, werden in der Timeseries-Datenbank InfluxDB abgespeichert. Die Eventdaten haben einen allgemeinen Teil, der bei allen Sensoren gleich ist, und einen Data Teil der jeweils unterschiedliche, sensorspezifische Informationen enthält.

Die Daten werden von Grafana [13] zur Visualisierung und von einem Workflow, zur Korrelation von Streaming-Telemetry- und Netflow-Daten, verwendet.

3.12 Netbox

Um das Netzwerk einem effektiven Troubleshooting unterziehen zu können, benötigt OATS eine «Source of Truth», welche den korrekten Netzwerkzustand abbildet. Dafür wird Netbox [14] verwendet, über welche die Netzwerkdaten per UI in eine PSQL-Datenbank [15] eingepflegt werden können.

Die Daten der Netbox werden von OATS in den Troubleshooting-Workflows verwendet, um z.B. den Nachbarn eines Interfaces auf einem Netzwerkgerät zu ermitteln.



3.13 Grafana

Um ein attraktives und relevantes Dashboard zu haben, werden Daten aus der InfluxDB von Grafana abgefragt, welche diese zusammenfasst und in Metriken darstellt.

Dies erlaubt einen schnellen Überblick über das System und ein einfaches Monitoring der Services.

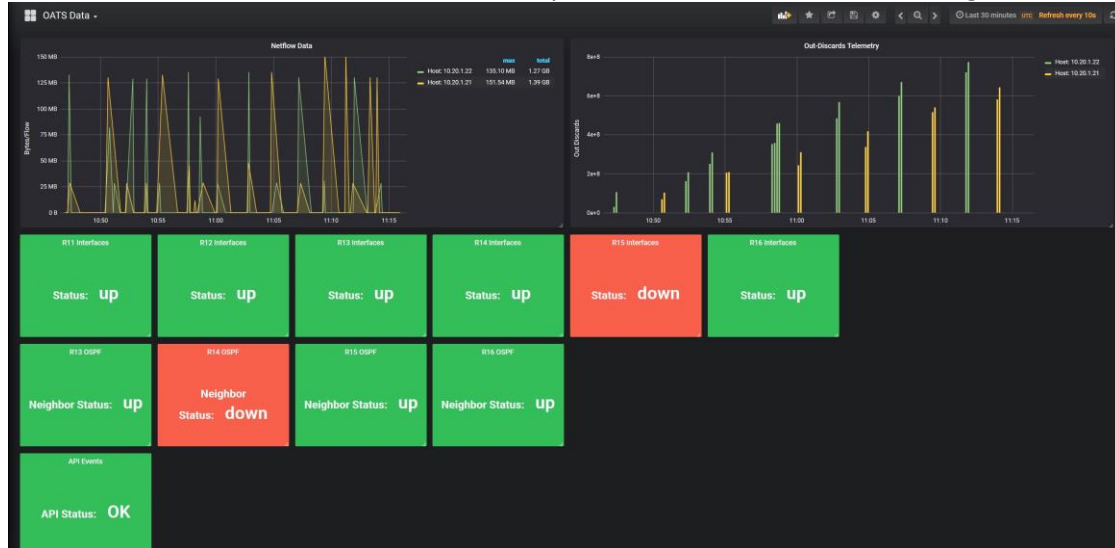


Abbildung 6: OATS-Grafana

4. Ergebnis

Durch die Kombination von verschiedenen Open Source Tools und Technologien wurde somit ein System entwickelt, mit welchem es möglich ist, Probleme in Netzwerken automatisch zu erkennen und zu behandeln. Um aufzuzeigen, was mit den beiden Sensoren neu möglich ist, wurde ein Troubleshooting-Workflow umgesetzt, welcher sich den Streaming-Telemetry und Netflow-Daten bedient.

Zusammen mit dem bereits bestehenden Sensor Syslog ist es so möglich eine grosse Bandbreite von möglichen Fehlern im Netzwerk zu überwachen.

Eine grosse Verbesserung wurde durch das neue Daten-Konzept mit Kafka und InfluxDB erreicht. Dank diesen beiden Tools ist es theoretisch möglich OATS so zu skalieren, dass es auch mit grossen Datenmengen umgehen kann.

Mit dem Hintergedanken der Skalierung wurde noch ein Problem mit einem der Tools, welches in der Studienarbeit ausgewählt wurde, bemerkt. So ist Salt an sich gut skalierfähig, ist aber nicht für automatisches Troubleshooting ausgelegt und stösst bei seinem Zeitverhalten und bei vielen gleichzeitigen Troubleshooting-Jobs an seine Grenzen.

Weiter wurde mit Grafana eine graphische Oberfläche zur Anzeige der erhaltenen Daten hinzugefügt und das Erstellen von neuen Troubleshooting Workflows wurde vereinfacht.



5. Ausblick

Das Projektteam zeigt sich mit OATS zufrieden und ist der Meinung es wäre bereit in einem echten Netzwerk getestet zu werden.

Dort könnten unter anderem noch folgende Funktionalitäten hinzugefügt werden:

- **Testen der Skalierfähigkeit:** Durch die gewählten Tools (Kafka, InfluxDB) ist OATS theoretisch in der Lage unendlich horizontal zu skalieren, indem OATs auf mehrere Server verteilt wird.
- **Machine-Learning:** Machine-Learning setzt reale Daten voraus, welche während der Entwicklung von OATS nicht verfügbar waren. Diese sind in einem realen Umfeld vorhanden und deswegen würde sich das integrieren von Machine-Learning lohnen.
- **Interaktives Troubleshooting:** Im Moment wird das Troubleshooting durchgeführt ohne dass manuell eingegriffen werden kann. Manchmal wäre es aber durchaus sinnvoll, dass sich OATS von einem Menschen bestätigen lässt, ob eine Änderung sinnvoll ist (→ Chat-Ops).
- **Fehlerklassifizierung:** Zurzeit wird in OATS angenommen, dass wenn ein Fehler eintritt, dass dieser schwer genug wiegt, dass er mittels automatischem Troubleshooting behandelt werden muss. In der Realität gibt es aber eine grosse Bandbreite an Fehlern, welche alle unterschiedlich dringend sind. Die Grundlage dafür wurde in OATS schon gesetzt, muss aber noch ausgearbeitet werden.
- **Ausbauen der API:** Die API ist zurzeit noch simpel gestaltet und ist ausbaufähig. Eine gute API würde auch die Grundlage für ein GUI für die Benutzer von OATS darstellen.

6. Danksagung

Das Thema «Automated Troubleshooting» ist uns bereits während der vorangegangenen Studienarbeit ans Herz gewachsen und so sind wir unserem Betreuer Prof. Beat Stettler dankbar, dass wir diese Arbeit fortführen durften.

Obwohl wir im letzten vergangenen Jahr viel über Netzwerke gelernt haben, existierten immer noch einige Wissenslücken in diesem Bereich. Deswegen möchten wir uns noch bei Urs Baumann bedanken, der uns bei einigen Problemen im Umgang mit den Netzwerkgeräten geholfen hat. Auch für die Hilfestellung die er und Prof. Laurent Metzger, zum Erfassen von möglichen Problemen in Netzwerken, leisteten, möchten wir uns bedanken.

Zuletzt möchten wir uns noch bei der Open Source Community bedanken, ohne die eine solche Arbeit unmöglich wäre.



II. Software-Engineering Dokumente



Projekt: Automatisiertes Troubleshooting

Raphael Jöhl
Nico Vinzens



1. Domainanalyse

Änderungsgeschichte

Datum	Version	Änderung	Autor
19.09.2017	1.0	Erstellen des Dokuments	nv
04.10.2017	1.1	Technologieanalyse	nv
05.10.2017	1.2	Technologieanalyse	rj
06.10.2017	1.3	Technologieanalyse, Problemanalyse, Deviceanalyse	rj
07.10.2017	1.4	Problemanalyse	rj
15.12.2017	2.0	Korrektur vor Abgabe	nv
02.03.2018	3.0	Übernahme Dokument in BA	nv
03.03.2018	3.1	Verbesserungen aufgrund von Feedback	nv
15.03.2018	3.2	Erweiterungsanalyse für die BA	rj, nv
20.03.2018	3.3	Überarbeitung nach Sitzung	rj, nv
03.06.2018	3.4	Überarbeitung vor Abgabe	rj, nv



1.1 Einführung

1.1.1 Zweck

Dieses Dokument wurde erstellt, um einen Überblick über die Domain, in der im Rahmen dieses Projekts gearbeitet wird, zu erlangen.

Das automatisierte Troubleshooting in einem Netzwerk ist ein sehr komplexes Thema, um präzise und sinnvolle Anforderungen an unser System zu erstellen und definieren, analysieren wir in diesem Dokument die Technologien die uns zur Verfügung stehen, Probleme die in einem Netzwerk auftreten und Geräte mit welchen wir in einem Netzwerk arbeiten. Zudem wird analysiert, wie das während der Studienarbeit entwickelte System noch erweitert werden kann.

Als Erstes kommt eine Technologieanalyse in der beschrieben wird, mit welchen Tools und Technologien Informationen über Geräte und das Netzwerk beschafft werden können.

In der Problemanalyse folgt eine Auflistung an möglicher Probleme die in einem System auftreten können und wie diese detektiert werden können. Als Einführung folgt dazu zuerst eine kurze Erläuterung über die vier unteren OSI Layer, auf welche wir uns in den möglichen Problemen konzentrieren möchten.

Danach werden in der Deviceanalyse die möglichen Geräte, mit welchen wir in einem Netzwerk arbeiten, beschrieben und wieso hier Probleme auftreten.

Zum Schluss folgt die Erweiterungsanalyse, in denen wir uns mit möglichen Themen für die Erweiterung des bestehenden Systems befassen.

Anmerkung: die Kapitel «Technologieanalyse», «Problemanalyse» und «Deviceanalyse» wurden bereits während der Studienarbeit erstellt und in der Bachelorarbeit überarbeitet und ergänzt, während das Kapitel «Erweiterungsanalyse» neu in der Bachelorarbeit dazu gekommen ist.

1.1.2 Gültigkeitsbereich

Das Dokument ist gültig während der Entwicklung des automatisierten Troubleshooting Systems OATS.



1.2 Technologieanalyse

In diesem Abschnitt werden verschiedene Technologien beschrieben und analysiert, die im Netzwerkumfeld benutzt werden können, um Informationen über den Status des Netzwerks zu erhalten.

Die vorgestellten Technologien liefern entweder Informationen über den Zustand des Netzwerks (Ist das Netz funktionsfähig?) oder Daten über den durch Benutzer verursachten Verkehr im Netz.

Deswegen werden die Technologien jeweils im letzten Unterkapitel («Anwendung beim Troubleshooting») in **Zustandssensor** oder **Daten-Sensor** kategorisiert.

Der grösste Unterschied der gelieferten Daten der beiden Sensoren ist, dass diejenigen von Zustandssensor nicht mit historischen Daten verglichen werden müssen, während die Daten von Daten-Sensoren noch in einem Zwischenschritt mit historischen Daten verglichen werden müssen, um Aussagen über das Netzwerk machen zu können.

1.2.1 Ping

Ping ist ein Protokoll welches es uns erlaubt innerhalb eines IP Netzwerks zu überprüfen ob ein Host verfügbar ist. Ein ping misst die Round Trip Time zwischen zwei Hosts. Ping verwendet Internet Control Message Protocol (ICMP/ICMP6) Echo Request Pakete um beim Ziel eine Anfrage zu platzieren, auf die dieser mit einer ICMP Echo Response antwortet.

[16]

Options

Die Optionen von ping unterscheiden sich je nach Implementation innerhalb des Systems. Die Optionen verfügbar unter Windows 10 sind:

Options:

Option	Beschreibung
-t	Pingt das Ziel solange, bis es gestoppt wird. Um eine Statistik anzuzeigen und weiterzufahren kann man "Control-Break" eingeben, um den ping zu beenden kann man „Control-C“ eingeben.
-a	Wandelt Adressen in Hostnamen um
-n <Anzahl>	Die Anzahl der zu übertragenden echo requests
-l <Grösse>	Die Grösse des Payloads eines Pakets
-f	Setzt das „nicht Fragmentieren“ Flag, nur in IPv4
-i	Spezifiziert die Time to Live
-r <Anzahl>	Zeichnet die Route für die spezifizierte Anzahl Sprünge auf, nur in IPv4
-s <Anzahl>	Gibt die Zeit an für die erreichte Anzahl Sprünge, nur in IPv4
-j <host-list>	Loose Route entlang der angegebenen Hosts, nur in IPv4
-k <host-list>	Strenge Route entlang der angegebenen Hosts, nur in IPv4
-w <timeout>	Wie lange in Millisekunden auf jede Antwort gewartet werden muss
-S <src-addr>	Falls eine alternative Quelle verwendet werden soll
-4	Erzwingt IPv4
-6	Erzwingt IPv6



Beispiel Ping

Hier wurde eine Web Adresse fünf Mal „angepingt“.

```
U:\>ping -n 5 hsr.ch

Pinging hsr.ch [152.96.12.22] with 32 bytes of data:
Reply from 152.96.12.22: bytes=32 time<1ms TTL=127
Reply from 152.96.12.22: bytes=32 time=1ms TTL=127
Reply from 152.96.12.22: bytes=32 time=1ms TTL=127
Reply from 152.96.12.22: bytes=32 time=1ms TTL=127
Reply from 152.96.12.22: bytes=32 time=1ms TTL=127

Ping statistics for 152.96.12.22:
    Packets: Sent = 5, Received = 5, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

Abbildung 7: 3: ping -n5 auf hsr.ch

Format

IP Datagram				
	Bits 0–7	Bits 8–15	Bits 16–23	Bits 24–31
IPv4 Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		flags and offset	
	Time To Live (TTL)	Protocol	Header Checksum	
	Source IP address			
	Destination IP address			
ICMP Header (8 bytes)	Type of message	Code	Checksum	
	Identifier		Sequence Number	
ICMP Payload (optional)	Payload Data			



IP Datagram						
	Bits 0–3	Bits 4–7	Bits 8–11	Bits 12–15	Bits 16–23	Bits 24–31
IPv6 Header (40 bytes)	Version	Traffic Class		Flow Label		
	Payload Length				Next Header	Hop Limit
	Source Address					
	Destination Address					
ICMP6 Header (8 bytes)	Type of message		Code		Checksum	
	Header Data					
ICMP6 Payload (<i>optional</i>)	Payload Data					

In einem regulären IPv4 Header ist „Protocol“ auf 1 gesetzt für ICMP und „Type of Service“ auf 0.
In einem regulären IPv6 Header ist das Feld „Next Header“ auf 58 gesetzt.

Der ICMP Header beinhaltet das 8-Bit lange „Type of message“ Feld, das 8-Bit lange „Code“ Feld und das „Checksum“ Feld ist 16 Bit lang und gibt an ob die Summe der Einsen innerhalb der ICMP Nachricht korrekt ist.

Das Header Data Feld welches hier entweder Echo requests oder replies sind, bestehen dadurch aus dem 16 Bit „Identifizier“ und der 16 Bit „Sequence Number“.

Der ICMP Payload kann selber definiert werden, jedoch muss der Payload mit den Headern kleiner sein als die Maximum Transmission Unit oder man riskiert eine Fragmentierung.

Echo Reply und Request

Die oben beschriebenen Identifizier und Sequenz Nummer Felder dienen dazu bei einem Echo Request die dazu gehörenden Antwort (Echo Reply) zu finden. Die Details des Identifiers sind je nach Implementation unterschiedlich.

Ein Echo Reply ist die Antwort auf einen Echo Request, sie ist für alle Hosts zwingend und muss den genauen erhaltenen Payload angeben.

Payload

Grundsätzlich wird Payload bei Angabe eines solchen mit ASCII Zeichen gefüllt. Der Payload kann ausserdem separat einen Timestamp und eine Sequenznummer enthalten. Dies ermöglicht es einem Ping die Round Trip Zeit stateless zu ermitteln ohne die Zeit der Übertragung jedes Pakets zu ermitteln.

Anwendung beim Troubleshooting

Ping ist ein **Zustandssensor** und wird beim Troubleshooting vor allem verwendet um die Konnektivität zu Geräten zu überprüfen. Ist ein Gerät über Ping nicht erreichbar, so lässt sich das vorhandene Problem auf die unteren Netzwerklayer eingrenzen (L1-L3). Dies könnte z.B. auf ein physikalisches Problem (Kabelbruch) oder auf ein falsches IP-Subnetting hinweisen.



1.2.2 Traceroute

Traceroute ist ein Netzwerk-Diagnose-Tool welches uns erlaubt die Route und Verzögerungen von Paketen über ein IP Netzwerk zu ermitteln. Der Verlauf der Route wird zusammen mit der Round Trip Time der Pakete welche man von jedem nachfolgendem Host erhalten hat gespeichert. Die Summe der durchschnittlichen Zeit zwischen jedem Schritt ergibt die Zeit welche insgesamt benötigt wurde um die Verbindung herzustellen. Traceroute sendet jeweils drei Pakete, wenn diese mehr als zwei Mal verloren gingen ist die Verbindung verloren und die Route kann nicht evaluiert werden. Anders als bei Ping werden jeweils die Zeit für die einzelnen Schritte aufgeschrieben und nicht nur die insgesamt benötigte Zeit.

[17], [18]

Implementation

Die Implementation von traceroute unterscheidet sich bei Unix basierten Systemen und Windows Systemen. Unter Unix werden per default eine Sequenz von UDP oder TCP Paketen gesendet mit der Option ICMP Echo Request. Unter Windows wird traceroute über ICMP Echo Requests gesendet.

Um herauszufinden wie viele Schritte bis zum Ziel nötig sind verwendet traceroute den Time to Live Wert. Erhält ein Router ein Paket reduziert er die TTL um eins, falls die TTL auf null ist, verwirft er das Paket und sendet ein ICMP Time Exceeded Paket zurück. Erhält der Host von welchem der traceroute gestartet wurde eine solche Nachricht erhöht er die TTL des Pakets um eins um den nächsten Schritt des Pfades herauszufinden. Somit kann der Sender Schritt für Schritt eine Liste mit allen Routern und deren Verbindungszeit erstellen. Dies geht so lange weiter bis das Paket das Ziel erreicht hat. Das eigentliche Ziel des traceroutes sendet dann eine ICMP Echo Reply Nachricht und signalisiert dass das Ziel erreicht wurde.

Der Sender hat jeweils eine Zeitgrenze wie lange er auf die Reply eines Schrittes wartet. Wird diese überschritten zeigt er in der Liste einen Stern. Da das Internet Protocol nicht erzwingt dass die Pakete immer die gleiche Route verwenden, kann es sein, dass die Liste mit den Hosts auf dem Weg des traceroutes unterschiedlich ist wie die schlussendlich überquerten Hosts.

In einem Netzwerk welches sowohl Windows als auch Unix System hat, müssen in einer eventuell vorhandenen Firewall mehrere Protokolle erlaubt sein, ansonsten funktioniert traceroute nicht.

Es gibt unterschiedliche Arten von traceroute, so gibt es zum Beispiel auch tcptraceroute und Layer 4 traceroute. Ein weiteres Werkzeug ist Pathping welche die Funktionalitäten von Ping und traceroute verbindet.



Beispiel Traceroute

```
U:\>tracert google.com

Tracing route to google.com [172.217.19.14]
over a maximum of 30 hops:

  1  <1 ms    <1 ms    <1 ms    router-192.hsr.ch [152.96.192.1]
  2   3 ms     2 ms     1 ms     hsrpr1.hsr.ch [152.96.2.49]
  3   1 ms     22 ms    3 ms     152.96.2.33
  4   1 ms     1 ms     1 ms     swiAG1-10GE-0-0-1-1.switch.ch [130.59.37.13]
  5   1 ms     1 ms     1 ms     swiEZ1-10GE-2-1.switch.ch [130.59.38.2]
  6   2 ms     2 ms     2 ms     swiEZ3-B3.switch.ch [130.59.36.34]
  7   2 ms     2 ms     2 ms     swiZH1-100GE-0-1-0-0.switch.ch [130.59.38.110]
  8   2 ms     2 ms     2 ms     72.14.195.4
  9   2 ms     2 ms     2 ms     209.85.244.127
 10   2 ms     2 ms     2 ms     zrh04s08-in-f14.1e100.net [172.217.19.14]

Trace complete.
```

Abbildung 8: tracert auf google.com

In Abbildung 2 sieht man ein tracert-Beispiel für google.com unter Windows. Rechts sieht man die IP-adressen und falls vorhanden die Namen der traversierten Router.

Optionen

Nachfolgend die zu Verfügung stehenden Optionen für traceroute unter Unix Systemen.

Option	Beschreibung
-4, -6	Zwingt traceroute dazu IPv4 oder IPv6 zu verwenden, per default wird das Protokol automatisch gewählt
-I	Benutzte ICMP ECHO für die Pakete, wie tracert unter Windows
-T	Benutzte TCP SYN für die Pakete
-U	Benutzte UDP Datagramme für die Pakete, dies ist der Default unter Unix
-F	Setzt das „Nicht Fragmentieren“ Bit
-ffirst_<ttl>	Spezifiziert mit welcher TTL gestartet wird, Default ist 1
-mmax_<ttl>	Spezifiziert die maximale Nummer der Schritte, Default ist 30
-iinterface	Spezifiziert über welches Interface traceroute gesendet werden soll, per Default wird dieses nach der Routing Table gewählt
-Nsqueries	Spezifiziert die Anzahl der Pakete die gleichzeitig gesendet werden

Anwendung beim Troubleshooting

Durch den **Zustandssensor** traceroute lässt sich genau nachvollziehen, welchen Weg ein Paket durch das Netz nimmt. So lassen sich mögliche Routing-Probleme (ineffizientes Routing, Loops) erkennen. Da bei jedem Hop die Latenz gemessen wird, ist es auch möglich, etwaige Bottlenecks im Netz zu identifizieren.



1.2.3 SNMP – Simple Network Management Protocol

Ein SNMP-Management System beinhaltet [19]:

- mehrere Knoten, die alle über eine SNMP-Instanz verfügen, welche einen *command responder* und *notification originator applications* beinhalten (auch Agenten genannt);
- mindestens eine SNMP-Instanz mit einem *command generator* und/oder *notification receiver applications* (auch Manager genannt) und
- ein Management-Protokoll, um Informationen zwischen den SNMP-Entitäten auszutauschen.

SNMP-Manager überwachen und steuern Geräte, auf welchen SNMP-Agenten aufgesetzt sind. Solche Geräte sind typischerweise Server, Router etc.

Funktionsweise

Die Agenten, welche direkt auf den zu überwachenden Geräten laufen, sind in der Lage den Zustand eines Netzwerkgerätes zu erfassen und auch selbst Einstellungen vorzunehmen oder Aktionen auszulösen. Die Agenten kommunizieren mit dem Manager über ein Netzwerk, dazu gibt es sechs verschiedene Datenpakete die gesendet werden können:

Datenpaket	Beschreibung
GET-REQUEST	Zum Anfordern eines Management-Datensatzes
GETNEXT-REQUEST	Um den nachfolgenden Datensatz abzurufen (für Tabellen)
GETBULK	Um eine angegebene Anzahl an Datensätzen gleichzeitig abzurufen
SET-REQUEST	Um einen oder mehrere Datensätze eines Geräts zu verändern
GET-RESPONSE	Antwort auf eines der vorherigen Pakete
TRAP	Nachricht eines Agenten an den Manager, dass ein Ereignis eingetreten ist. Kann benutzt werden um eine Fehlfunktion oder das nicht erfolgreiche Ausführen eines SET-REQUESTS zu melden.

GET-Pakete werden vom Manager an einen Agenten geschickt, um Informationen über das jeweilige Gerät anzufordern.

Mit einem SET-REQUEST-Paket kann ein Manager Werte bei einem Agenten verändern. Die erfolgreiche Änderung bestätigt der Agent mit einem GET-RESPONSE-Paket.

Stellt ein Agent bei der Überwachung eines Systems einen Fehler fest, so kann er dies dem Manager mithilfe eines Trap-Pakets unaufgefordert melden. Trap-Pakete werden vom Manager nicht bestätigt.

Zum Übermitteln der Pakete wird das Transportprotokoll UDP verwendet. Der Agent empfängt SNMP-Requests auf Port 161, während der Manager Trap-Pakete auf Port 162 empfängt.

Die Datenbasis des jeweiligen Geräts ist in der MIB (Management Information Base) hinterlegt.

Paket Aufbau

SNMP-Paket-Header		PDU-Header				PDU-Body			
Versionsnummer	Community Name	Pakettyp (Get, GetNext, ...)	RequestID	Fehlerstatus	Fehler-Index	Variable Binding 1	Variable Binding 2	...	Variable Binding n

Abbildung 9: SNMP-Paket Aufbau [9]

SNMP-Paket-Header

Versionsnummer: SNMPv1, SNMPv2 oder SNMPv3

Community Name: Zum Erteilen von Rechten werden Communities zugewiesen (z.B. private & public)

PDU-Header (Nicht-Trap-Pakete)

Pakettyp: z.B. GET-REQUEST, GETBULK etc.

Request-ID: Um Antwortpakete den vorausgegangenen Anfragen zuordnen zu können



Fehlerstatus: wird bei Antwortpaketen verwendet um mitzuteilen, weshalb eine Anfrage nicht bearbeitet werden konnte.

Fehlerindex: gibt im Fehlerfall an, beim wievielten Datensatz der Fehler aufgetreten ist

PDU-Header (Trap-Pakete)

PDU-Header					
Pakettyp (Trap)	OID des Gerätes, das den Trap generiert hat	IP-Adresse des Absenders	allgemeine TrapID	firmenspezifische TrapID	Zeitpunkt des Auftretens des Trap-Ereignisses

Abbildung 10: Trap-PDU-Header

OID des Geräts: herstellerspezifische OID (Object Identifier), gibt an um was für ein Gerät es sich handelt

Absender-IP: IP-Adresse des Absenders

Allgemeine Trap-ID:

Bezeichnung	Trap-ID
coldStart	0
warmStart	1
linkDown	2
linkUp	3
authenticationFailure	4
egpNeighborLoss	5
enterpriseSpecific	6

Bei ID 6 (enterpriseSpecific), wird die ID des firmenspezifischen Traps im Feld danach übertragen. Um zeitliches Ordnen zu ermöglichen, wird zusätzlich noch die Zeitangabe übermittelt, bei der das Ereignis ausgelöst wurde.

PDU-Body

Beinhaltet die eigentlichen Werte. Jeder Wert wird in einer sogenannten Variable Binding übertragen. Zu solch einer Variable Binding gehören jeweils dessen OID und der Wert selber.

Vor- und Nachteile

Vorteile:

- SNMP ist ein offener Standard, d.h. herstellerunabhängig
- Die meisten Netzwerkgeräte unterstützen SNMP

Nachteile:

- Begrenzte Funktionalität: Cisco-Geräte unterstützen zum Beispiel nur ca. 90 verschiedene Traps (ggü. 6500 bei syslog) [20]

Anwendung beim Troubleshooting

SNMP kann als eventbasierter **Zustandssensor** im Netzwerk eingesetzt werden und dient somit als Auslöser für Troubleshooting-Fälle.



1.2.4 NETCONF – Network Configuration Protocol

Das NETCONF-Protokoll bietet einen Mechanismus um Netzwerkgeräte zu managen, Konfigurationsdaten abzurufen und um neue Konfigurationsdaten hochzuladen und zu manipulieren. Das Protokoll erlaubt es einem Gerät eine API zu diesen Zwecken anzubieten. Applikationen können diese API benutzen um auf Gerätekonfigurationen zuzugreifen und sie zu ändern.

NETCONF wurde entworfen, um das Verwalten von Netzwerkgeräten per SSH zu abstrahieren. Es greift selbst auch per SSH auf die Geräte zu, allerdings sind die Daten strukturiert.

Um dies zu erreichen benutzt NETCONF eine XML-basierte Datenencodierung für die Konfigurationsdaten und auch die Protokoll-Nachrichten.

Das NETCONF-Protokoll arbeitet auf einem RPC-Layer, d.h. ein Host kann extern definierte Methodenaufrufe ausüben.

[21]

Protokoll-Layer

Das NETCONF-Protokoll ist konzeptuell in 4 Layer aufgeteilt:

- Content
- Operations
- Messages
- Secure Transport

Content-Layer

Der Inhalt von NETCONF-Operations ist XML und bezieht sich grösstenteils auf Netzwerk-Management. Mittlerweile werden auch das JSON-Format und YANG unterstützt.

Operations-Layer

Das Grundprotokoll von NETCONF erlaubt folgende Operationen:

Operation	Beschreibung
<get>	Ruft die aktuell laufende Konfiguration und den Gerätestatus ab
<get-config>	Ruft eine bestimmte Konfiguration ab
<edit-config>	Manipuliert eine Konfiguration
<copy-config>	Kopiert eine Konfiguration an einen anderen Ort
<delete-config>	Löscht eine Konfiguration
<lock>	Blockiert die Konfiguration eines Geräts
<unlock>	Gibt die Konfiguration eines Geräts frei
<close-session>	Terminiert eine laufende NETCONF-Session (kontrolliert)
<kill-session>	Erzwingt die Terminierung einer NETCONF-Session

Die Grundoperationen von NETCONF sind erweiterbar und werden direkt per Server und Client ausgehandelt.

Später wurde noch eine Möglichkeit zum Abonnieren von Events hinzugefügt. Dies kann mittels des <create-subscription> Befehls erreicht werden, welcher das Erstellen von *Subscriptions* in Echtzeit erlaubt. Damit werden Benachrichtigungen asynchron mittels des <notification> Konstrukts verschickt.

Messages

Die NETCONF-Messages bieten einen transportunabhängigen Weg um *RPC invocations* (<rpc> messages), *RPC results* (<rpc-reply> messages) und *even notifications* (<notification> messages) zu kodieren.



Vor – und Nachteile

Vorteile:

- Wurde für Automation ausgelegt (Subscription <> Notifications)
- Modellgetrieben und transaktionsbasiert (mittels YANG & RPC)

Nachteile:

- Relativ neu, deswegen noch nicht so weit verbreitet
- Benötigt eine dauerhafte Verbindung zwischen Netzwerkgerät, welches die Notifications liefert und dem Server, welcher sie erhalten möchte.

Anwendung beim Troubleshooting

NETCONF hat zwei Anwendungsfälle beim Troubleshooting. Zum einen kann es eingesetzt werden, um bei einem aktiven Troubleshooting-Fall Gerätekonfigurationen abzufragen und zu ändern. Zum anderen kann es auch als **Daten-Sensor** eingesetzt werden, indem der Subscription/Notification Mechanismus verwendet wird.

1.2.5 YANG

YANG ist eine Datenmodellierungssprache und wird benutzt um Konfigurations- und Statusdaten zu beschreiben. YANG ist baumstrukturiert, d.h. die Konfigurationsdaten sind aufgebaut wie ein Baum und die Daten können komplexe Typen wie Listen oder Vereinigungsmengen beinhalten [4].

YANG Terminologie (RFC 6020)

[22]

augment	Adds new schema nodes to a previously defined schema node
container	An interior data node that exists in at most one instance in the data tree. A container has no value, but rather a set of child nodes.
data model	A data model describes how data is represented and accessed.
data node	A node in the schema tree that can be instantiated in a data tree. One of container, leaf, leaf-list, list, and anyxml.
data tree	The instantiated tree of configuration and state data on a device.
derived type	A type that is derived from a built-in type (such as uint32), or another derived type.
device deviation	A failure of the device to implement the module faithfully.
grouping	A reusable set of schema nodes, which may be used locally in the module, in modules that include it, and by other modules that import from it. The grouping statement is not a data definition statement and, as such, does not define any nodes in the schema tree.
identifier	Used to identify different kinds of YANG items by name.
leaf	A data node that exists in at most one instance in the data tree. A leaf has a value but no child nodes.
leaf-list	Like the leaf node but defines a set of uniquely identifiable nodes rather than a single node. Each node has a value but no child nodes.
list	An interior data node that may exist in multiple instances in the data tree. A list has no value, but rather a set of child nodes.
module	A YANG module defines a hierarchy of nodes that can be used for NETCONF-based operations. With its definitions and the definitions it imports or includes from elsewhere, a module is self-contained and “compilable”.
state-data	The additional data on a system that is not configuration data such as read-only status information and collected statistics [RFC4741].



YANG Beispiel

Folgendes Beispiel wurde von napalm-automation.net/yang-for-dummies/ entnommen:

```
// module name
module napalm-star-wars {

    // boilerplate
    yang-version "1";
    namespace "https://napalm-yang.readthedocs.io/napalm-star-wars";

    prefix "napalm-star-wars";

    // identity to unequivocally identify the faction an individual belongs to
    identity AFFILIATION {
        description "To which group someone belongs to";
    }

    identity EMPIRE {
        base AFFILIATION;
        description "Affiliated to the empire";
    }

    identity REBEL_ALLIANCE {
        base AFFILIATION;
        description "Affiliated to the rebel alliance";
    }

    // new type to enforce correctness of the data
    typedef age {
        type uint16 {
            range 1..2000;
        }
    }

    // this grouping will all the personal data we will assign to individuals
    grouping personal-data {
        leaf name {
            type string;
        }
        leaf age {
            type age;
        }
        leaf affiliation {
            type identityref {
                base napalm-star-wars:AFFILIATION;
            }
        }
    }

    // this is the root object defined by the model
    container universe {
        list individual {
            // identify each individual by using the name as key
            key "name";

            // each individual will have the elements defined in the grouping
            // uses personal-data;
        }
    }
}
```

Abbildung 11: YANG Modelldefinition



Hier wird anhand eines Beispiels aufgezeigt, wie man mit YANG Personen im Star-Wars Universum beschreiben kann:

- <Identity AFFILIATION> ist eine erfundene *identity* um Personen eindeutig einer Fraktion zuteilen zu können. Es wurden zwei Typen bestimmt: EMPIRE & REBELS. Alle Personen die mit diesem Modell dargestellt werden, müssen der einen oder der anderen Fraktion angehören.
- <age> beschreibt das Alter der Personen. Uint16 ist ein selber definierter Typ, um dem Alter eine Einheit geben zu können.
- <grouping personal-data> teilt dem Personenmodell die *leafs* zu, also Felder, die jede Person haben muss. Hier sind das die oben beschriebenen Felder <age> und <identity>.
- <container universe> stellt den Container dar, in dem alle Modelle beinhaltet sein müssen (→die „Wurzel“ des Baumobjekts). Jedes Modell innerhalb des Containers verfügt über einen Namen und über die Felder die in <grouping personal data> definiert wurden.

Mit diesem Modell kann man nun Personen des Star-Wars Universums beschreiben:

```
{
  "universe": {
    "individual": {
      "Obi-wan Kenobi": {
        "affiliation": "REBEL_ALLIANCE",
        "age": 57,
        "name": "Obi-wan Kenobi"
      },
      "Luke Skywalker": {
        "affiliation": "REBEL_ALLIANCE",
        "age": 19,
        "name": "Luke Skywalker"
      },
      "Darth Vader": {
        "affiliation": "EMPIRE",
        "age": 42,
        "name": "Darth Vader"
      },
      "Yoda": {
        "affiliation": "REBEL_ALLIANCE",
        "age": 896,
        "name": "Yoda"
      }
    }
  }
}
```

Abbildung 12: Beispiel eines YANG Datensatzes

Es ist auch möglich andere Modelle zu importieren und selber zu erweitern:

```
// We import the old model
import napalm-star-wars { prefix napalm-star-wars; }

// New identity based off the old AFFILIATION
identity MERCENARY {
  base napalm-star-wars:AFFILIATION;
  description "Friend for money";
}
```

Abbildung 13: Importieren und Erweitern eines bestehenden Modells



Anwendung beim Troubleshooting

YANG ist in Netzwerken eine wichtige Modellierungssprache um Gerätekonfigurationen zu beschreiben. So kann z.B. ein Router als root und seine Interfaces (und alle dazugehörigen Informationen) als leafs dargestellt werden. YANG ist somit kein Sensor, sondern wird benutzt, um Sensordaten in einer wohldefinierten Form zu modellieren und zu übertragen. So kommt YANG unter anderem bei Cisco Streaming-Telemetry, NETCONF und napalm (siehe Architektur-Abschnitt) zum Einsatz.

1.2.6 Syslog

Protokoll zum zentralen Sammeln von Informationen in einem Netzwerkkumfeld. Auszug aus dem Abstract des RFC 5424 [23]:

„This document describes the syslog protocol, which is used to convey event notification messages. This protocol utilizes a layered architecture, which allows the use of any number of transport protocols for transmission of syslog messages. It also provides a message format that allows vendor specific extensions to be provided in a structured way.“

Ein Server dient als zentrale Sammelstelle der Syslog Meldungen. Somit kann ein eventbasierter Troubleshooting-Prozess ausgelöst werden.

Message-Format

Eine Syslog Meldung beinhaltet einen *facility code* und einen *severity level*. Die Syslog Software fügt der Meldung noch zusätzliche Informationen wie eine Prozess-ID, ein Zeitstempel und Hostname oder IP des Geräts, hinzu.

Der Facility Code beschreibt den Programmtyp, der die Meldung erstellt hat. Es existieren folgende *facilities*:

Facility Code	Keyword	Description
0	kern	kernel messages
1	user	user-level messages
2	mail	mail system
3	daemon	system daemons
4	auth	security/authorization messages
5	syslog	messages generated internally by syslogd
6	lpr	line printer subsystem
7	news	network news system
8	uucp	UUCP subsystem
9		clock daemon
10	authpriv	security/authorization messages
11	ftp	FTP daemon
12	-	NTP subsystem
13	-	log audit
14	-	log alert
15	cron	scheduling daemon
16-23	local0-7	local use 0-7



Der Severity Level zeigt die Schwere des aufgetretenen Fehlers an. Es existieren folgende *severity level*:

Value	Severity	Keyword	Description
0	Emergency	emerg	System is unusable
1	Alert	alert	Action must be taken immediately
2	Critical	crit	Critical conditions, eg. hard device errors
3	Error	err	Error conditions
4	Warning	warning	Warning conditions
5	Notice	notice	Normal but significant conditions
6	Informational	info	Informational messages
7	Debug	debug	Debug-level messages

Die Bedeutung der verschiedenen Severity Levels kann von der Applikation, die das Syslog Protokoll benutzt, selbst bestimmt werden.

Konfiguration

Syslog wird über eine zentrale Konfigurationsdatei gesteuert und heisst meistens */etc/syslog.conf*. In der Datei wird beschrieben, wie Meldungen zu schreiben sind und wie diese über das Netzwerk übertragen werden [1]. Die Datei hat folgende Form:

```
#important kernel-level warnings
Kern.warning                /dev/tty10
```

Links steht ein *facility-severity* Paar, welches angibt was für eine Message erzeugt werden soll. Rechts steht, wohin die Messages geloggt werden sollen.

Will man eine zentrale Sammelstelle für Syslog Meldungen definieren so geht das folgendermassen:

```
#central logging server gets a copy of all the warnings
*.warn                      @192.168.1.1
```

So werden alle auf diesem Gerät generierten warnings an die IP-Adresse 192.168.1.1 kopiert und geschickt.

Vor- und Nachteile

Vorteile:

- Grösserer Funktionsumfang als z.B. SNMP-Traps (bis zu 6000 unterschiedliche Meldungen bei Cisco)
- So können mittels Syslog sehr viele Probleme die in einem Netzwerk auftauchen, unterschieden werden

Nachteile:

- Syslog Meldungen sind plaintext-basiert, d.h. haben keine vorbestimmte Form (können aber durch verschiedene Tools aber in Objekte/Modelle umgewandelt werden)
- Unterstützt nur PUSH Nachrichten, Syslog Konfigurationen können nicht durch das Syslog Protokoll selbst remote geändert werden

Anwendung beim Troubleshooting

Syslog kann wie auch SNMP als eventbasierter **Zustandssensor** im Netz eingesetzt werden um Troubleshooting-Fälle auszulösen.



1.2.7 Netflow

Netflow ist ein Feature welches die Möglichkeit bietet Network Traffic zu sammeln sobald er ein Interface betritt oder verlässt. Die gesammelten Informationen werden in UDP Datagrammen exportiert. Ursprünglich war Netflow eine Cisco Technik, sie wurde aber von vielen Herstellern übernommen. Netflow ist wie SNMP ein Messverfahren, d.h. es werden Informationen über den Verkehr gesammelt aber keine Änderungen vorgenommen. Router und Switches welche Netflow unterstützen können auf allen Interfaces auf denen Netflow aktiviert ist IP Traffic Informationen sammeln [24] [25].

Je nach Implementation des Herstellers unterscheiden sich die Informationen die in einem Network Flow enthalten sind. Unter Cisco beinhaltet ein Paket folgendes:

- IP Source Adresse
- IP Destination Adresse
- Source Port
- Destination Port
- Layer 3 Protocol
- Type/Class of Service
- Router/Switch Interface

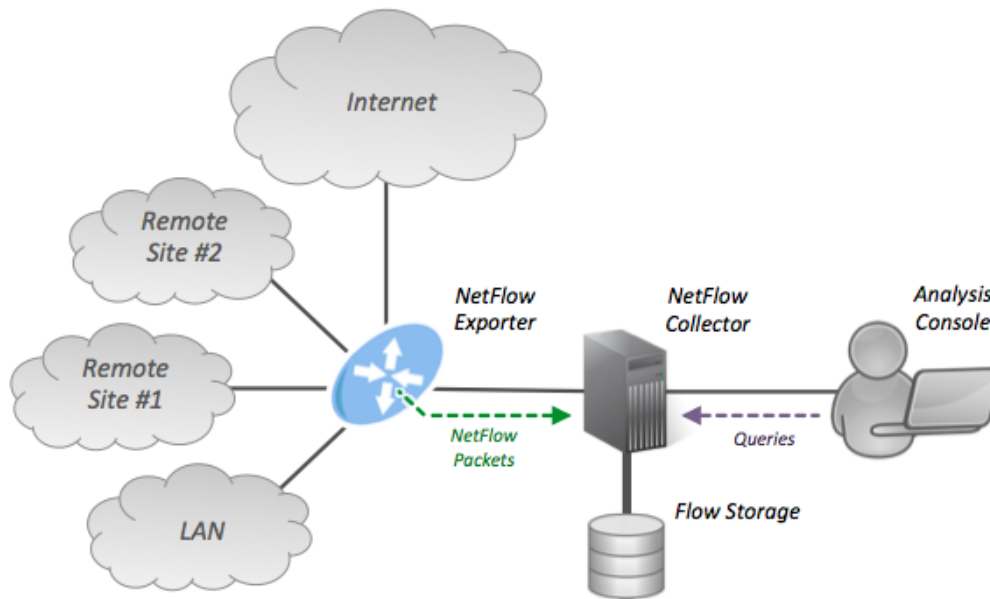
Pakete die diese Informationen gemeinsam haben können als Flow zusammengefasst werden. Diese Flow-Aufzeichnung beinhaltet weitere Informationen wie

- Input Interface Index von SNMP, ifIndex falls IF-MIB verwendet wurde
- Output Interface Index oder null falls das Paket gedroppt wurde
- Timestamps des Starts und Ende des Flows
- Anzahl Bytes und Pakete in einem Flow
- Layer 3 Header
- Source & destination IP addresses
- ICMP Type und Code
- IP Protokoll
- Type of Service (ToS) value
- Source und destination Portnummern für TCP, UDP, SCTP
- Für TCP Flows die Gemeinsamkeiten aller TCP Flags
- Layer 3 Routing Informationen, IP Adresse des next hop auf der Route

Die Analyse dieser Flow Daten dient dazu sich ein Bild über den traffic Flow und Volumen eines Netzwerks zu erstellen.



Funktionsweise



Ein

Abbildung 14: NetFlow Funktionsweise [9]

Netflow Setup besteht meistens aus drei Komponenten welche an ein Netzwerk angeschlossen werden.

Flow Exporter	Sammelt Pakete aus einem Netzwerk und bündelt diese in Flows, welche dann zum Collector exportiert werden.
Flow Collector	Die Exportierten Flow Pakete werden vom Collector erhalten und gespeichert.
Analysis application	Um die erhaltenen Flow Daten analysieren zu können braucht es eine Basis mit der man Informationen vergleichen kann und daraus Schlüsse ziehen kann.

Versionen und Alternativen

Netflow wurde ursprünglich von Cisco erstellt und weiterentwickelt. Diverse weitere Hersteller haben ihr eigenes Flow Monitoring entwickelt. Die weitverbreitetste Alternative ist sFlow. Welche jedoch einige Unterschiede zu Netflow aufweist, so erlaubt sFlow das exportieren von Paket Daten Chunks und Interface Counters.

Grundsätzlich unterstützen die meisten Netzwerk Geräte Netflow Versionen 5 und 9, jedoch gibt es auch hier Unterschiede.

Vor- und Nachteile

Vorteile:

- Eine grosse Anzahl Pakete können zusammengefasst werden und mit Einbezug des Kontexts analysiert werden

Nachteile:

- zusätzliche Processing-Power nötig für jedes Paket
- Viele verschiedene Versionen und Support dieser sind nicht für alle möglichen Geräte gewährleistet



Anwendung beim Troubleshooting

Netflow ist ein **Daten-Sensor** und sendet kontinuierlich Daten zum Collector. Der Collector muss diese Daten erst noch verarbeiten und evaluieren, bevor er entscheiden kann, ob ein Problem vorliegt.

1.2.8 Netzwerk-Sniffing

Ein Netzwerk-Sniffer überwacht den Datenfluss innerhalb eines Netzwerks in Echtzeit. Sniffer können in Software oder Hardware umgesetzt sein [26].

Funktionsweise

Sniffer erstellen Kopien von Paketen, ohne dass die originalen Pakete umgeleitet oder geändert werden. Dies erlaubt es einer Applikation oder einem Benutzer den Inhalt des Pakets genau zu untersuchen. Sofern die Pakete nicht verschlüsselt sind, kann so der gesamte Netzwerk-Protokoll-Stack des Pakets, von L2 bis L7(OSI), überprüft werden. Beim TCP-Protokoll kann so z.B. eine genaue Statistik erstellt werden, wie oft eine Verbindung neu aufgebaut werden muss und wie viele Pakete unterwegs gedroppt werden.

So existieren diverse Tools, mit denen in Echtzeit Pakete analysiert werden. Sniffer Pro z.B. kann so den Durchsatz, die Auslastung, Errors/s messen und im Problemfall Alarme (→events) auslösen.

Vor- und Nachteile

Vorteile:

- Erlaubt es Fehler zu entdecken, die sonst unerkannt bleiben
- Erlaubt Durchsatzmessungen (→Erkennung langsamer Verbindungen)

Nachteile:

- Es ist schwierig in Echtzeit den Datenverkehr zu analysieren und in kurzer Zeit darauf zu schließen wo der Fehler liegt.
- Packet-Inspection basiert meistens auf der Suche nach gewissen Stichworten (z.B. /etc/shadow), dies ist nützlich bei der Überwachung eines Netzes, aber nur wenig geeignet um Troubleshooting-Prozesse auszulösen.

Anwendung beim Troubleshooting

Ein Netzwerk-Sniffer könnte als Daten-Sensor im Netz eingesetzt werden. Allerdings müssten dafür die oben genannten Nachteile zuerst gelöst werden.

1.2.9 IP SLA - Internet protocol service level agreement

Internet Protocol Service Level Agreement oder kurz IP SLA ist ein Protokoll von Cisco IOS, welches es erlaubt Informationen über Network Performance in Echtzeit zu sammeln. Dieses Monitoring wird über ein aktives Verfahren erstellt, d.h. Ein IP SLA Router hat die Fähigkeit, Traffic zu generieren und diesen direkt zu überprüfen [27].

Die SLAs sammeln Daten über Antwortzeit, Latenz, Jitter, Paketverluste, Connectivity, Server oder Webseiten Antwortzeit und Downtime, Pfad und Paket Sequenzierung. Mit diesen Werten ist es möglich eine grundsätzliche Information über die Netzwerkperformance zu sammeln. Zusätzlich ist es so möglich die Quality of Service überprüfen und man sieht wo die Performance zu niedrig ist. Innerhalb eines Network Management Systems ist es zusätzlich möglich eine Visualisierung der Grenzwerte und Überschreitungen dieser zu erhalten.



IP SLA benötigt mindestens zwei Geräte in einem zu überprüfenden Netzwerk, den IP SLA Router welcher Traffic erzeugt, der zweite ist der Responder, dieser kann von einem beliebigen Hersteller sein.

Anwendung

Hier eine Liste mit Messungen die via IP SLA erstellt werden können sowie die verwendeten Technologien dazu.

Cisco IOS IP SLAs Operation	Messungen
UDP Jitter	Misst round-trip Verzögerung, one-way Verzögerung, one-way Jitter, one-way Paketverlust und Connectivity-Testing von Netzwerken welche UDP traffic besitzen wie etwa Voice Channels.
ICMP Path Jitter	Misst hop-by-hop Jitter, Paketverlust, and Verzögerung
UDP Jitter für VoIP	Misst round-trip Verzögerung, one-way Verzögerung, one-way Jitter, and one-way Paketverlust von VoIP traffic.
UDP Echo	Misst round-trip Verzögerung von UDP traffic.
ICMP Echo	Misst round-trip Verzögerung für den ganzen Pfad.
ICMP Path Echo	Misst round-trip Verzögerung and hop-by-hop round-trip Verzögerung.
HTTP	Misst round-trip time um eine Webseite abzurufen.
TCP Connect	Misst die Zeit die benötigt wird eine TCP Verbindung herzustellen.
FTP	Misst round-trip time um ein File zu senden.
Dynamic Host Configuration Protocol (DHCP)	Misst round-trip time die benötigt wird eine IP Adresse von einem DHCP Server zu erhalten
Domain Name System (DNS)	Misst DNS lookup Zeit.
Data Link Switching Plus (DLSw+)	Misst peer tunnel Antwortzeit.
Frame Relay	Misst circuit Verfügbarkeit, round-trip Verzögerung, and frame delivery Rate. Diese Operation hat keine SNMP Unterstützung.

Vor- und Nachteile

Vorteil:

- Überprüfen des Netzwerks von einem zentralen Punkt.
- Viele Konfigurationsmöglichkeiten.
- Erreichbar durch SNMP
- Ermöglicht das Auslösen von Events bei z.B. zu hoher Latenz auf einem Link

Nachteil:

- IP SLA ist eine Funktion von Cisco Geräten, mindestens ein Gerät im zu überprüfenden Netzwerk muss ein Cisco Gerät sein und IP SLA fähig sein.
- Das Monitoring funktioniert über aktiv generierten Traffic der zusätzlich das Netzwerk belastet.
- Die Überprüfungen müssen sehr genau spezifiziert werden.
- Die Dokumentation ist wenig übersichtlich, so ist z.B. nicht genau spezifiziert, wie auf die gesammelten Daten zugegriffen werden kann.



Anwendung beim Troubleshooting

IP SLA kann als **Daten-Sensor** eingesetzt werden um die Netzwerkperformance zu analysieren. Allerdings müssten die generierten Daten noch durch eine Logik überprüft werden, um zu erkennen, wann und wo ein Problem vorliegt.

1.2.10 SSH – Secure Shell

SSH ist ein Netzwerk Protokoll welches die Funktionalität bietet eine sichere Verbindung über ein unsicheres Netzwerk zu erstellen und somit sichere Netzwerk Operationen auszuführen, hierbei wird ein Client –Server Ansatz verwendet.

In einem Netzwerk bietet es uns die Möglichkeit remote Command-Line Funktionen auszuführen. So können von einem zentralen Punkt Gerätekonfigurationen überprüft und manipuliert werden.

1.2.11 Streaming-Telemetry

Streaming-Telemetry ist ein neuer Ansatz zur Netzwerküberwachung in welchem kontinuierlich Daten von Geräten in inkrementellen Updates gestreamt werden (Push anstatt Pull). Eine zentrale Sammelstelle kann dabei angeben, für welche Daten sie sich interessiert und erhält nur diese Daten. Anders als bei Netflow wird das Netz kaum belastet, da z.B. nur alle 10 Sekunden Daten übermittelt werden.

Telemetry Modell

Folgende Darstellung zeigt die Funktionsweise von Streaming-Telemetry auf [28]:

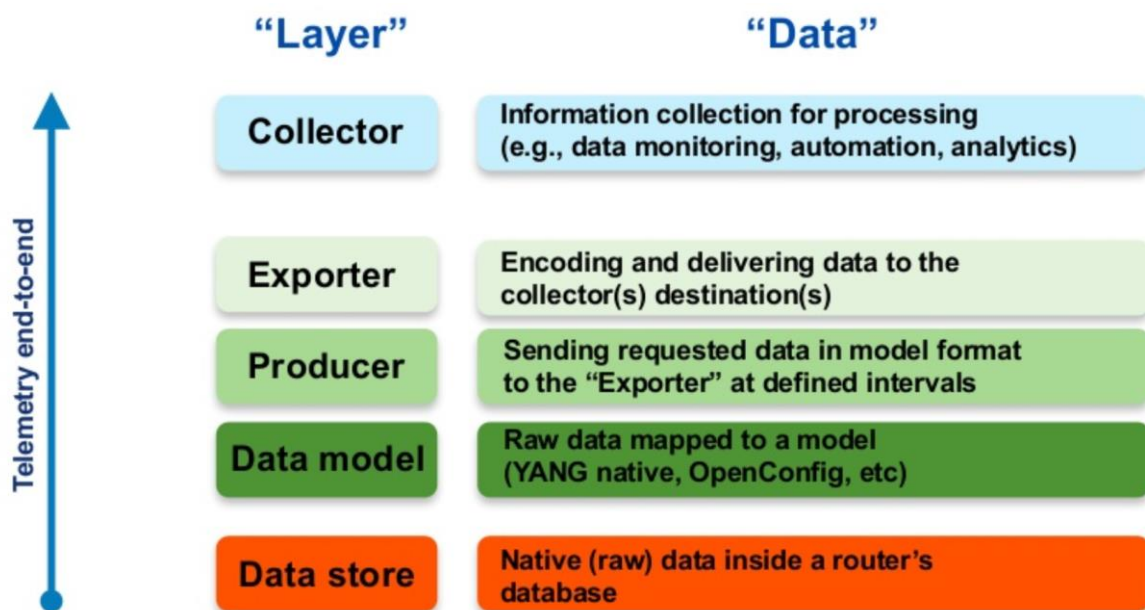


Abbildung 15: Telemetry-Modell

Data store: Beinhaltet die Daten die gestreamt werden.

Data model: Das Modell auf das die rohen Daten aus dem Data store gemappt werden.

Producer: Das Netzwerkgerät, welches die Daten streamt.

Exporter: Die Übertragungsweise der Daten. Kann klassisch über TCP/UDP passieren oder mittels dem relativ neuen gRPC [29]. Das Encoding ist entweder Google Protocol Buffers (GPB [30]) oder JSON.

Collector: Der Server der auf einem Port auf die Streaming-Telemetry Daten horcht und die Daten verarbeitet/speichert.



Cisco Streaming-Telemetry

Bei Cisco IOS-XR Routern werden die Daten in Form von YANG-Modellen abgebildet und z.B. über gRPC übertragen. Die nachfolgende Abbildung illustriert, wie ein solches Modell aussehen könnte [31]:

```
"RootOper": {  
  "InfraStatistics": {  
    "GigabitEthernet0/0/0/0": {  
      "Latest": {  
        "GenericCounters": {  
          "OutputQueueDrops": 0,  
          "LastDiscontinuityTime": 1449091544,  
          "InputIgnoredPackets": 0,  
          "PacketsReceived": 40881,  
          "OutputDrops": 0,  
          "UnknownProtocolPacketsReceived": 0,  
          "RuntPacketsReceived": 0,  
          "CRCErrors": 0,  
          "SecondsSinceLastClearCounters": 0,  
          "CarrierTransitions": 0,  
          "MulticastPacketsSent": 26784,  
          "BytesSent": 43668872,  
          "ThrottledPacketsReceived": 0,  
          "Applique": 0,  
          "FramingErrorsReceived": 0,  
          "GiantPacketsReceived": 0,  
          "OutputUnderruns": 0,  
          "OutputErrors": 0,  
          "BroadcastPacketsReceived": 0,  
          "OutputBuffersSwappedOut": 0,  
          "Resets": 0,  
          "SecondsSincePacketSent": 0,  
          "InputAborts": 0,  
          "InputOverruns": 0,  
          "InputQueueDrops": 0,  
          "InputDrops": 0,  
          "AvailabilityFlag": 0,  
          "MulticastPacketsReceived": 26775,  
          "SecondsSincePacketReceived": 0,  
          "OutputBufferFailures": 0,  
          "BytesReceived": 43635706,  
          "ParityPacketsReceived": 0,  
          "BroadcastPacketsSent": 2,  
          "LastDataTime": 1449320856,  
          "InputErrors": 0,  
          "PacketsSent": 40904  
        }...  
      }  
    }  
  }  
}
```



Abbildung 16: Interface Yang-Model

Wie man sieht enthält dieses Modell bereits einige für das Troubleshooting nützliche Felder wie «CRCErrors» oder «GiantPacketsReceived». Diese Daten können dynamisch und in Echtzeit im Code verarbeitet werden.

Eine Übersicht über die verfügbaren Yang-Modelle für Cisco Streaming-Telemetry ist unter [32] verfügbar. Darin sieht man, dass unter anderem Modelle für Netflow und IP SLA existieren. Somit wäre es theoretisch möglich auch diese Daten über Streaming-Telemetry zu übermitteln und so das Datenformat für alle Sensoren gleich zu halten.



IOS-XE vs. IOS-XR

Bei der Recherche zu Cisco Streaming-Telemetry wurde festgestellt, dass sich die Konfiguration von Streaming-Telemetry bei IOS-XE grundsätzlich von der von IOS-XR unterscheidet.

So muss bei IOS-XR lediglich folgendes konfiguriert werden [33]:

```
telemetry model-driven
  destination-group DGroup1
    address family ipv4 172.30.8.4 port 5432
    encoding self-describing-gpb
    protocol tcp
  !
!
sensor-group SGroup1
  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics,
!
subscription Sub1
  sensor-group-id SGroup1 sample-interval 30000
  destination-id DGroup1
```

Abbildung 17: IOX-XR Telemetry-Konfiguration

So muss der Server auf dem angegebenen Port auf die Daten warten und diese verarbeiten.

Bei IOS-XE läuft alles über NETCONF-Sessions. So muss zuerst eine NETCONF-Session zwischen dem Server welcher die Daten möchte und dem IOS-XE-Gerät etabliert werden. Danach muss der Server mittels XML-RPC-Call angeben, welches Modell er abonnieren möchte. Das heisst, dass für die Dauer in der man die Streaming-Telemetry-Daten erhalten möchte, eine NETCONF-Session aufrechterhalten muss. Dies ist eigentlich nicht der Sinn von Streaming-Telemetry, da dies mehr Pull- als Push darstellt.

Hat man die Wahl zwischen IOS-XR und IOS-XE muss man klar IOS-XR wählen, wenn man mit Streaming-Telemetry arbeiten möchte.



Streaming Telemetry vs. traditionelle Datenüberwachung

Folgende Darstellungen zeigen auf, wie sich Streaming-Telemetry von der traditionellen Netzwerküberwachung unterscheidet:

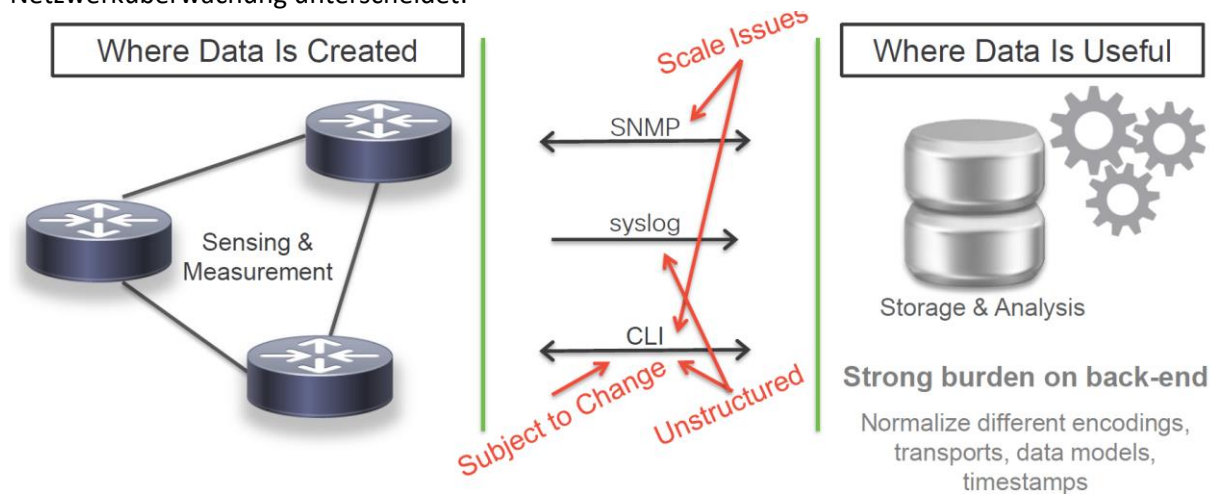


Abbildung 18: traditionelle Netzwerküberwachung

Bei der traditionellen Datenüberwachung muss viel Aufwand betrieben werden um die Daten in eine Form zu bringen in der sie dynamisch verarbeitet werden können (so wie auch OATS dies mit Syslog macht).

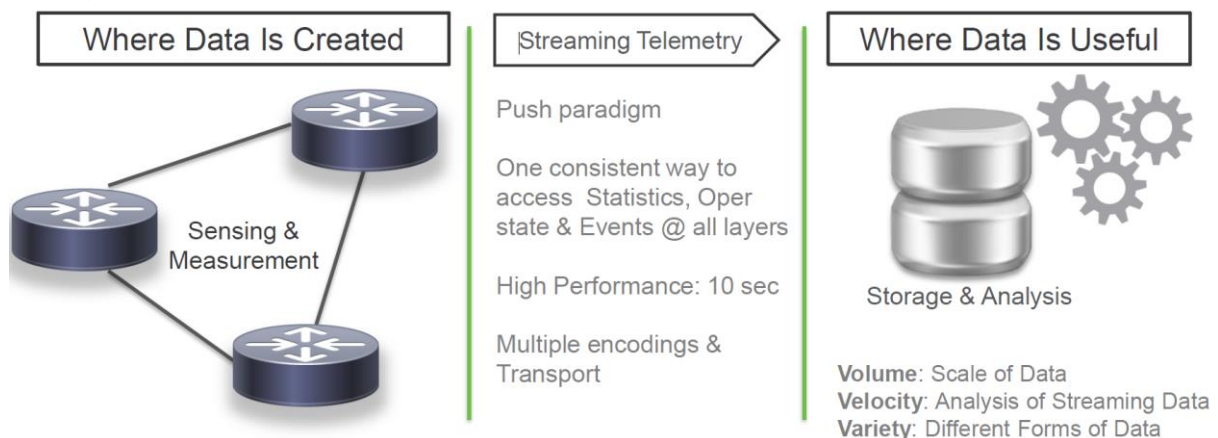


Abbildung 19: Netzwerküberwachung mittels Streaming-Telemetry

Streaming-Telemetry wurde für die dynamische Datenverarbeitung entwickelt, d.h. die Daten können direkt im Code ohne weitere Massnahmen analysiert werden. Zudem skaliert es um einiges besser als z.B. SNMP.

Eine offene Frage Bei Streaming-Telemetry ist die Verarbeitung der übertragenen Daten. Es wären unterschiedliche Ansätze denkbar, diese werden in Kapitel 1.5.1(Event-Korrelation) vorgestellt.

Anwendung beim Troubleshooting

Streaming-Telemetry ist in der Lage, die Funktionalität eines Zustandssensors mit derjenigen eines Daten-Sensors zu vereinigen. So können z.B. Daten über den Zustand eines Netzwerkgeräts gestreamt werden und gleichzeitig auch solche über den zurzeit generierten User-Traffic. Es ist sogar möglich, die Daten von anderen Sensoren (IP SLA, Netflow) auszulesen und kontinuierlich an einen Server zu streamen.



1.3 Problemanalyse

In einem Netzwerk können diverse Probleme auftauchen, im Rahmen dieser Analyse wollen wir uns auf die Probleme im Layer eins bis vier gemäss dem OSI-Modell konzentrieren. Hier eine Erläuterung der vier relevanten Layer [34], [35].

1.3.1 OSI-Modell

	Layer	Daten	Beschreibung
Media-Layer	4. Transport	Segment (TCP) Datagram (UDP)	Verlässliche Übertragung von Daten Segmenten zwischen zwei Punkten eines Netzwerks, inklusive Segmentierung, Bestätigung und Bündelung
	3. Netzwerk	Paket	Strukturierung und Verwaltung eines Netzwerks inclusive Adressierung, Routing und traffic Kontrolle
	2. Data Link	Frame	Verlässliche Übermittlung von Data Frames zwischen zwei Physisch verbundenen Punkten
Host-Layer	1. Physical	Bit	Übermittlung von raw bit Streams über eine Physisches Medium

Layer 1: Physical Layer

Der physische Layer spezifiziert die Elektrische und Physischen Eigenschaften der Datenverbindung. Er definiert die Verbindung eines Geräts und des physischen Übermittlungskanals, z.B. ein Kupferkabel ein Glasfaserkabel oder eine Radiofrequenzverbindung. Dazu gehören ausserdem das Layout der einzelnen Pins, die Stromspannung, der Wechselstromwiderstand, diverse Kabelspezifikationen, Signal Timing und weiter Charakteristiken welche für verbundenen Geräte benötigt werden.

Es ist verantwortlich für die Übermittlung und den Empfang von unstrukturierten Rohdaten über ein physisches Medium. Übertragungsmodi wie Simplex, Halbduplex und Full duplex werden hier definiert.

Der physische Layer bietet die Grundlage für alle darüber liegenden Layer. Funktioniert der Layer nicht, so ist kein Datenverkehr möglich

Auf dem physischen Layer gibt es die low-level Netzwerkgeräte, wie etwa Hubs, Netzwerkadapter, Glasfaserkonvertierer und Repeater. Der physische Layer interessiert sich nicht für Protokolle der höheren Layer.

Layer 2: Data Link Layer

Der Data Link Layer bietet Punkt zu Punkt Datentransfer, eine Verbindung zwischen zwei direkt verbundenen Punkten. Er detektiert und korrigiert eventuell vorkommende Fehler aus dem physischen Layer. Dieser Layer definiert Protokolle um Verbindungen zwischen zwei physisch verbundenen Punkten aufzubauen und zu beenden. Diese Protokolle können auch Flow Control durchführen.

Der IEEE 802 Standard welcher sich mit Netzwerkstandards für Layer 1 und 2 auseinandersetzt Definiert zwei Sublayer im Data Link Layer:

Der Media Access Control, MAC, Layer, dieser ist verantwortlich für die Kontrolle wie zwei Geräte im Netzwerk eine Verbindung zu einem Medium aufbauen und dessen Erlaubnis Daten zu übermitteln.



Der Logical Link Control, LLC, Layer ist verantwortlich für die Identifizierung und Verkapselungsfunktionen, sowie die Fehlerüberprüfung und Frame-Synchronisation.

Layer 3: Network Layer

Der Netzwerk Layer bietet die Funktionalität und Prozeduren um Datensequenzen von variabler Länge, auch Datagramme genannt, von einem Gerät zu einem anderen Gerät zu übermitteln, auch wenn diese in unterschiedlichen Netzwerken liegen.

Ein Netzwerk ist hier eine Verbindung, ein Netz, von verschiedenen Geräten, die alle eine eigene Adresse haben und es erlaubt den verbundenen Geräten Mitteilungen zu senden, solange man deren Adresse kennt. Das Netzwerk übernimmt das Senden der Nachricht über einen Pfad. Falls die Nachricht zu gross ist um einzeln übermitteln zu werden, bietet der Netzwerk Layer die Möglichkeit diese in Fragmente aufzuteilen, einzeln zu übermitteln und an einem anderen Punkt wieder zusammen zu führen. Ausserdem besteht die Möglichkeit Übermittlungsfehler zu signalisieren.

Die Nachrichtenübermittlung über den Netzwerk-Layer muss nicht zwingend zuverlässig sein. Ein Netzwerk Protokoll kann diese Funktionalität bieten, muss es aber nicht.

Zum Netzwerk Layer gehören Protokolle, welche Routing übernehmen, Gruppenverwaltung für Multicast, Netzwerk Layer Informationen, Fehler anzeigen, sowie Netzwerk Layer Adressverwaltung.

Layer 4: Transport Layer

Der Transport Layer bietet Funktionalitäten und Prozeduren um Daten Sequenzen von variabler Länge von einem Ursprungs- zu einem Ziel-Host zu übermitteln und die Quality of Service Funktionalität beizubehalten, die Hosts können über ein oder mehrere Netzwerke verteilt sein.

Der Transport Layer kontrolliert die Zuverlässigkeit einer beliebigen Verbindung, über Flow Control, Segmentierung und Fehler Kontrolle. Protokolle können Status und Verbindungs- orientiert sein. Segmente können verfolgt und bestätigt werden, dadurch kann man diese erneut übermitteln falls sie nicht ankommen. Der Transport Layer kann einzelne Nachrichten welche er von höheren Layer erhält in Pakete bündeln.

Die Hauptsächlich verwendeten Protokolle im Transport Layer sind UDP und TCP, welche unterschiedliche Eigenschaften und Verwendungszwecke haben.

1.3.2 Probleme

Da wir nun wissen welche Funktionen uns auf jedem Layer zur Verfügung stehen können wir eine Liste mit möglichen Problemen, welche wir in einem Netzwerk antreffen könnten, erstellen. Diese Liste wurde während der Studienarbeit zur ersten Übersicht der Probleme erstellt. Für eine Auflistung der in der Bachelorarbeit neu hinzugekommenen Probleme verweisen wir auf das Kapitel 1.3 (Erweiterungsanalyse).

Layer 1 Probleme

Power

- Geräte können ausgeschaltet werden oder ihre Power geht aus.
- **Detektieren:** Ein Interface ist nicht mehr verfügbar.

Disconnect – Interface

- Ein mit einem Interface verbundenes Gerät wird plötzlich entfernt.
- **Detektieren:** Ein Interface ist nicht mehr verfügbar.

Bitfehler



- Einzelne Bits können fehlerhaft sein, was zu grösseren Fehler oder sogar Paketverlust führen kann.
- **Detektieren:** Der CRC ist fehlerhaft.

Kabelprobleme

- Ein Kabel ist beschädigt, weshalb eine Verbindung unzuverlässig wird.
- **Detektieren:** Der CRC ist bei der gleichen Verbindung oft fehlerhaft.

Runts

- Ein Paket ist kleiner als die 64 Byte, wegen einer schlechten Verkabelung oder einem fehlerhaften elektronischen Interface. Auch Überbleibsel einer Paketkollision können als runt enden.
- **Detektieren:** Cisco Geräte haben unter dem „show interface“ Befehl einen Zähler für Runts.

Temperature, power circuit overload

- Physische Fehler die auftreten können wenn mit elektrischen Geräten gearbeitet wird.
- **Detektieren:** Volt- und Temperaturmessungen

Layer 2 Probleme

Giants

Ein Paket welches mehr als 1500 Bytes Payload hat.

Detektieren: Falsche und unterschiedliche MTU-Einstellungen der Geräte.

CRC Fehler

- Der “cyclic redundancy value” auf einem Switch oder Router entspricht nicht dem berechneten Wert.
- **Detektieren:** Geräte haben einen Zähler für wie viele CRC Fehler bemerkt werden.

Collisions

- Pakete auf einem Fullduplex oder Halbduplex Interface können Kollidieren, kann zu runs führen.
- Late collisions, sind eine Art der Kollision welche nach den ersten 64 Byte des Frames erfolgen.
- **Detektieren:** Kollisionszähler auf den Geräten überprüfen.

STP Fehler

Das Spanning Tree Protokoll ist zuständig für die Inter-Switch Kommunikation.

- Obwohl ein Switch in einen arbeitsfähigen Modus wechseln müsste bleibt er in einem blockierenden Zustand, “blocking” state.
- **Detektieren:** Abrufen der Zustandsinformation eines Switches.
- Nachdem ein weiterer Switch an das Netz angeschlossen wurde, kommt es zu einem Wechsel in der Root Bridge, dies kann zu einer massenhaften Sendung von BPDU Pakten führen.
- **Detektieren:** Unerwartet hoher Traffic, Priorität der Switches in den STP Einstellungen.
- Da STP Parallele Verbindungen blockiert um Loops zu verhindern, ist der effektive Durchsatz eines Netzwerks kleiner als der eigentlich mögliche.
- **Detektieren:** Durchsatzmessungen
- Grosse Layer 2 Netzwerke haben oft ineffiziente oder falsche Pfade welche aus der STP device priority resultieren.
- **Detektieren:** Einträge in der STP Priorität Liste überprüfen, Durchsatzmessungen
- Obwohl STP Loops verhindern sollte, kommt es zu einem Loop im Netzwerk.
- **Detektieren:** Unerwartet hoher Traffic, dasselbe Paket wird mehrmals gesendet.

Flapping MAC-Adresse (Morell, 2017):

- Ein Switch erhält auf unterschiedlichen Interfaces Pakete von der gleichen MAC-Adresse.



- STP funktioniert nicht korrekt auf einem VLAN.
- **Detektieren:** MAC-Adressen Tabelle eines Switches auslesen, viele Switches loggen, wenn sie ein Flapping detektieren.

Broadcast Sturm durch DHCP Discovery

- In einem Netz mit einem DHCP Server sendet dieser eine exzessive Menge an DHCP Pakete.
- **Detektieren:** Geräte erhalten DHCP Pakete die sie nicht angefragt haben

Duplex Mismatch

- Zwei miteinander verbundene Geräte haben unterschiedliche Duplex Einstellungen, was zu Paketverlusten und Kollisionen führt.
- **Detektieren:** Überprüfen der Duplexeinstellung auf den betroffenen Geräten

MAC Addressing Errors

- Doppelte MAC-Adressen in der MAC-Address Table eines Switches
- Ein Switch fügt eine MAC-Adresse eines Geräts nicht in seine MAC-Adressen Tabelle hinzu
- **Detektieren:** Überprüfen der MAC-Adress Tabelle

Ungültige ARP Einträge

- Ein Gerät fügt ungültige Einträge in seine ARP Tabelle ein, falls diese für längere Zeit bestehen bleiben, zB. Wenige Minuten, kann man in dieser Zeit den betreffenden Host nicht erreichen.
- **Detektieren:** Gedropte Pakete wegen ungültiger Adresse, ARP Tabellen Einträge.

Geräte im gleichen VLAN können nicht erreicht werden

- Switchports sind nicht im korrekten VLAN eingetragen
- Auf einem Trunk Port ist VLAN nicht erlaubt.
- **Detektieren:** Pakete werden beim Switch gedroppt, Interface/Port Einstellungen überprüfen.

Gerät kann andere Geräte im Netz nicht erreichen

- Die Verbindung zwischen zwei Switchen funktioniert nicht, da sie sich in einem anderen Switchport Modus befinden (access/trunk).
- **Detektieren:** Anzeigen der Switchport Einstellungen auf beiden Geräten.
- Die Verbindung zwischen zwei Switchen funktioniert nicht, da sie andere Protokolle für die Channel Aggregation verwenden, zB. PAgP und LACP.
- **Detektieren:** Anzeigen der Switchport Einstellungen auf beiden Geräten.
- Die Verbindung zwischen einem Switch und einem Router funktioniert nicht, da sie andere Protokolle für ihre Trunk Encapsulation verwenden.
- **Detektieren:** Anzeigen der Interface Trunk Einstellungen auf beiden Geräten.

Layer 3 Switch Probleme

- Ein Layer 3 Switch muss auf den Ports die mit dem Layer 3 Netzwerk interagieren Routing aktivieren, hat er aber nicht.
- **Detektieren:** Anzeigen der Switchporteinstellungen auf dem spezifischen Port.
- Ein Layer 3 Switch hat das IP Routing standardmässig deaktiviert, müsste dies aber aktivieren um mit dem Layer 3 Netz zu interagieren.
- **Detektieren:** Anzeigen der IP Interface Einstellungen auf dem Switch.

Port Security Violation

- Ein Gerät hat keine Verbindung ins Netz, da er vom Switch blockiert wird, dieser hat eine Port Security Violation gemeldet.
- **Detektieren:** Die erlaubte statische MAC Adresse in den Port Security Einstellungen ist fehlerhaft.

CDP Probleme

- Fehlerhafte Einträge in der CDP Neighbor Tabelle.
- **Detektieren:** Überprüfen der CDP Neighbor Tabelle.
- Geräte die CDP nicht unterstützen, fluten erhaltene CDP Pakete über das Netzwerk,
- **Detektieren:** Netzwerk wird mit CDP Paketen überflutet, hoher Traffic.



Layer 3 Probleme

Allgemeine Routing Probleme

- Suboptimale Routen, das verwendete Routing Protokoll wählt ungünstige oder suboptimale Routen für die Pakete, was zu einer allgemeinen Verlangsamung des Netzes führt.
- **Detektieren:** Pakete brauchen länger als erwartet.
- Routing Loops, Innerhalb des Routing gibt es einen Loop.
- **Detektieren:** Unerwartet hoher Traffic dasselbe Paket wird mehrmals gesendet

NAT

- Ein PC im Netz mit NAT hat kein Zugriff auf eine Internetseite.
- **Detektieren:** Der PC hat einen falschen Standard Gateway
- Ein PC im Netz mit NAT hat kein Zugriff auf eine Internetseite, falsche Inside und Outside Interfaces eingestellt. Die IP Adresse des PC liegt ausserhalb der Access List Range welche der Router eingestellt hat.
- **Detektieren:** Verifizieren der NAT Einstellungen und der Access Liste auf dem Router.

IPv6 Probleme

- Ein PC hat keinen Zugriff auf eine IPv6 Adresse im Internet, hat aber Zugriff auf den Router.
- **Detektieren:** Verifizieren ob Router Advertisement auf dem Router unterdrückt ist.
- Ein PC hat keinen Zugriff auf eine IPv6 Adresse im Internet, auch keinen Zugriff auf den Router. Es wird SLAAC verwendet.
- **Detektieren:** Überprüfen der Interface Einstellungen auf dem Router, SLAAC funktioniert nur wenn der IP Prefix auf /64 gesetzt ist.



OSPF

- Der OSPF-Prozess eines Routers friert ein
- **Detektieren:** Auf den OSPF-Neighbors laufen die dead timer aus.
- In Abbildung 11 wird der Cisco-Troubleshooting-Prozess für OSPF dargestellt:

[36]

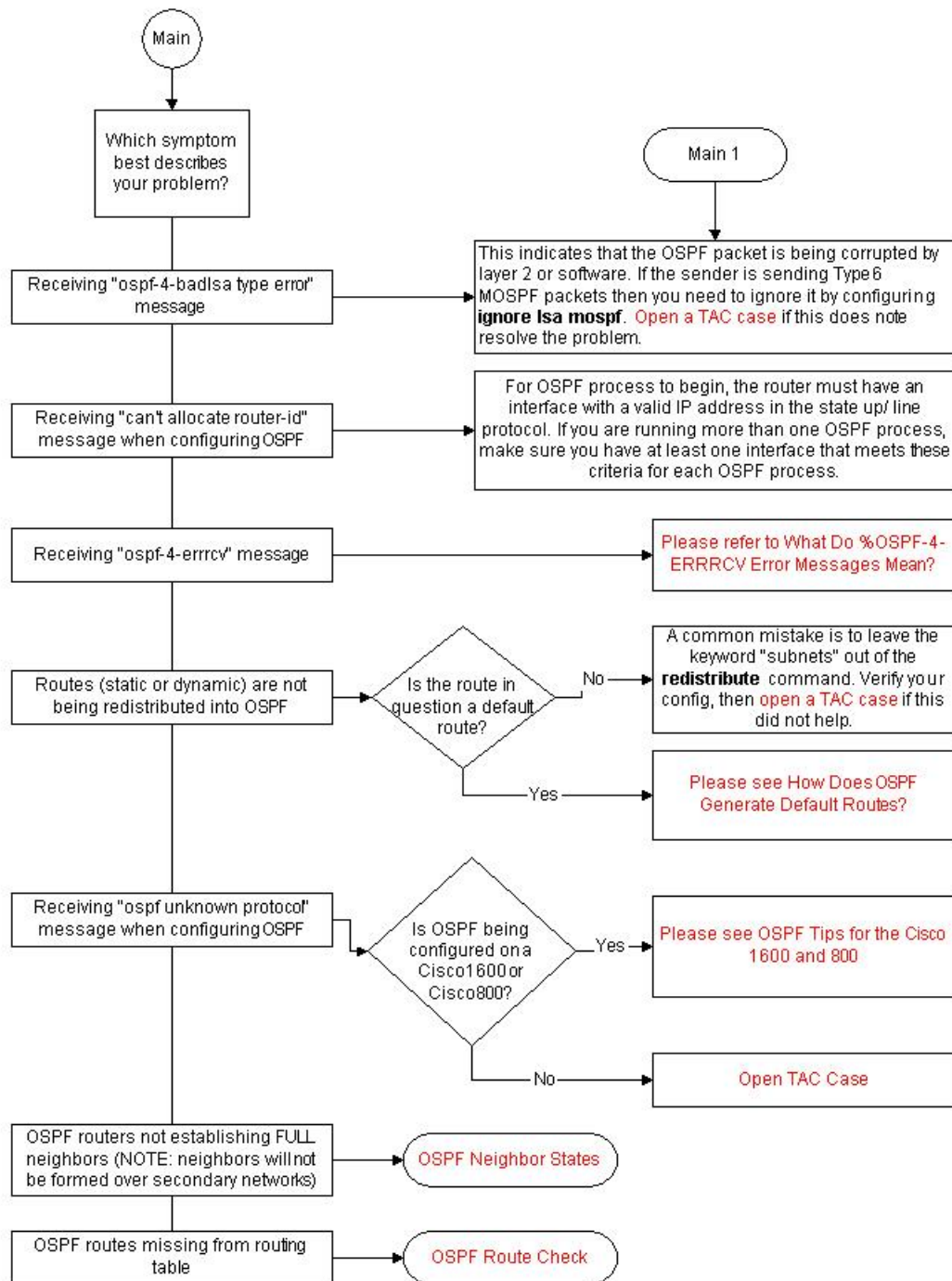


Abbildung 20: OSPF Troubleshooting-Prozess



EIGRP

- In Abbildung 12 wird der Cisco-Troubleshooting-Prozess für EIGRP dargestellt:

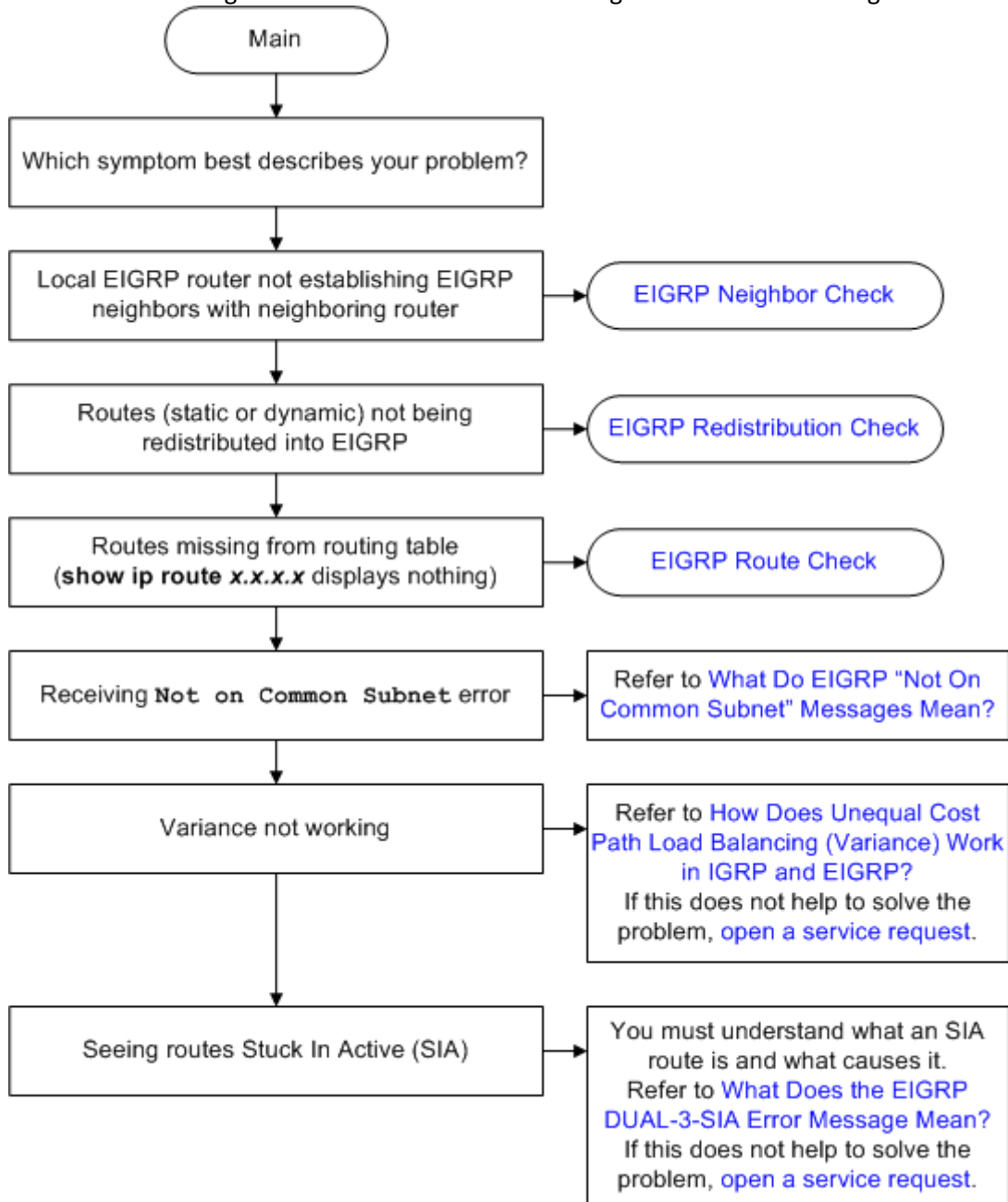


Abbildung 21: EIGRP Troubleshooting Prozess

Asymmetrisches Routing

- Ein Paket nimmt für eine beliebige Route zu einem Zielhost, jedoch eine andere Route für den Rückweg. Dies kann dazu führen, dass dieses Paket eine Firewall nicht durchqueren kann.
- Detektieren:** Nur Ein Weg des Traffic bleibt erhalten, überprüfen der Geräte, die Pakete wegen Rückwegpfaden dropen könnten.

Link Saturation

- Wenn ein Link mehr Daten übertragen soll als er eigentlich kann, 110Mbps auf einem 100Mbps Link, platziert er die übrigen Pakete in einem Buffer, welcher er leert sobald die



Sättigung zu Ende ist. Falls dies jedoch zu lange andauert und er den Buffer füllt, ist er gezwungen die Pakete zu dropen.

- **Detektieren:** Eine Erhöhung der Latenz, mehr Jitter und Erhöhung der gedropten Pakete

Layer 4 Probleme

Blockierte Ports

- Die Layer vier Protokolle TCP und UDP benötigen für ihre Kommunikation Ports, weshalb eine der häufigsten Probleme auf diesem Layer blockierte Ports sind, dies kann von einer Firewall oder direkt auf einer Access Liste eines Netzwerk Geräts sein.
- **Detektieren:** Überprüfen der Liste mit blockierten Ports, nachdem Pakete nicht ankommen.

Fragmentierung von grossen Datagrammen

- Wenn Datenstreams für die unteren Layer in Pakete umgewandelt werden, kann dies zu einigen Problemen führen.
- Falls bei TCP ein Datagramm in kleinere Fragmente aufgeteilt wird und eines dieser nicht ankommt müssen alle erneut gesendet werden. Dies kann zu Verlangsamungen im Netz führen.
- **Detektieren:** Langsames Netz, hohe Packet drop-rate.
- Falls mehrere fragmentierte Pakete in falscher Reihenfolge an eine Firewall kommen kann es sein dass die Firewall die Pakete droppt da ihr das initiale Paket fehlt.
- **Detektieren:** Firewall droppt Pakete, Verlangsamung des Netzwerks.

Weitere Probleme

HSRP Probleme

- Traffic läuft durch den falschen Switch, da die Priorität beim HSRP falsch ist, ist der Switch der aktiv sein sollte im Standby.
- **Detektieren:** Anzeigen der HSRP Prioritätseinstellungen auf dem Switch Interface.
- Traffic läuft durch den falschen Switch, der Router mit der höheren Priorität der aktiv sein müsste hat preemption deaktiviert.
- **Detektieren:** Anzeigen Standby Optionen auf dem Switch.
- HSRP-fail-over beim runterfahren des aktiven Switches funktioniert nicht korrekt.
- **Detektieren:** Anzeigen Standby Optionen auf dem Switch zeigt auch die Priorität decrement Option, falls der Switch failed muss diese so gewählt sein dass die Priorität kleiner wird als die des Standby Switches.

Telnet

- Ein PC hat keinen Telnet Zugriff auf einen anderen PC, sonst funktionieren aber alle Verbindungen.
- **Detektieren:** Verifizieren der Access Liste auf dem Router, ob dieser Telnet erlaubt.

DHCP

- Ein PC hat keinen Zugriff auf den DHCP Server der hinter einem Router liegt. Da dieser keine IP Helper-Adresse gewählt hat.
- **Detektieren:** Verifizieren der Adressen Informationen auf dem Router.

RIP

- Ein PC hat keinen Zugriff auf einen anderen Bereich des Netzwerks, es wird RIP verwendet. Ein Interface des Routers ist im passiven Modus, müsste aber aktiv sein. Was dazu führt dass er zwar RIP Informationen erhält aber keine sendet.
- **Detektieren:** Verifizieren der Router RIP Einstellungen. Überprüfen der gesendeten RIP Informationen.



BGP

- BGP läuft über TCP und ist ein Applikations-Layer Protokoll, wegen seiner Rolle im Routing haben wir es hier trotzdem erwähnt. In Abbildung 13 wird der Troubleshooting Guide von Cisco für BGP dargestellt:

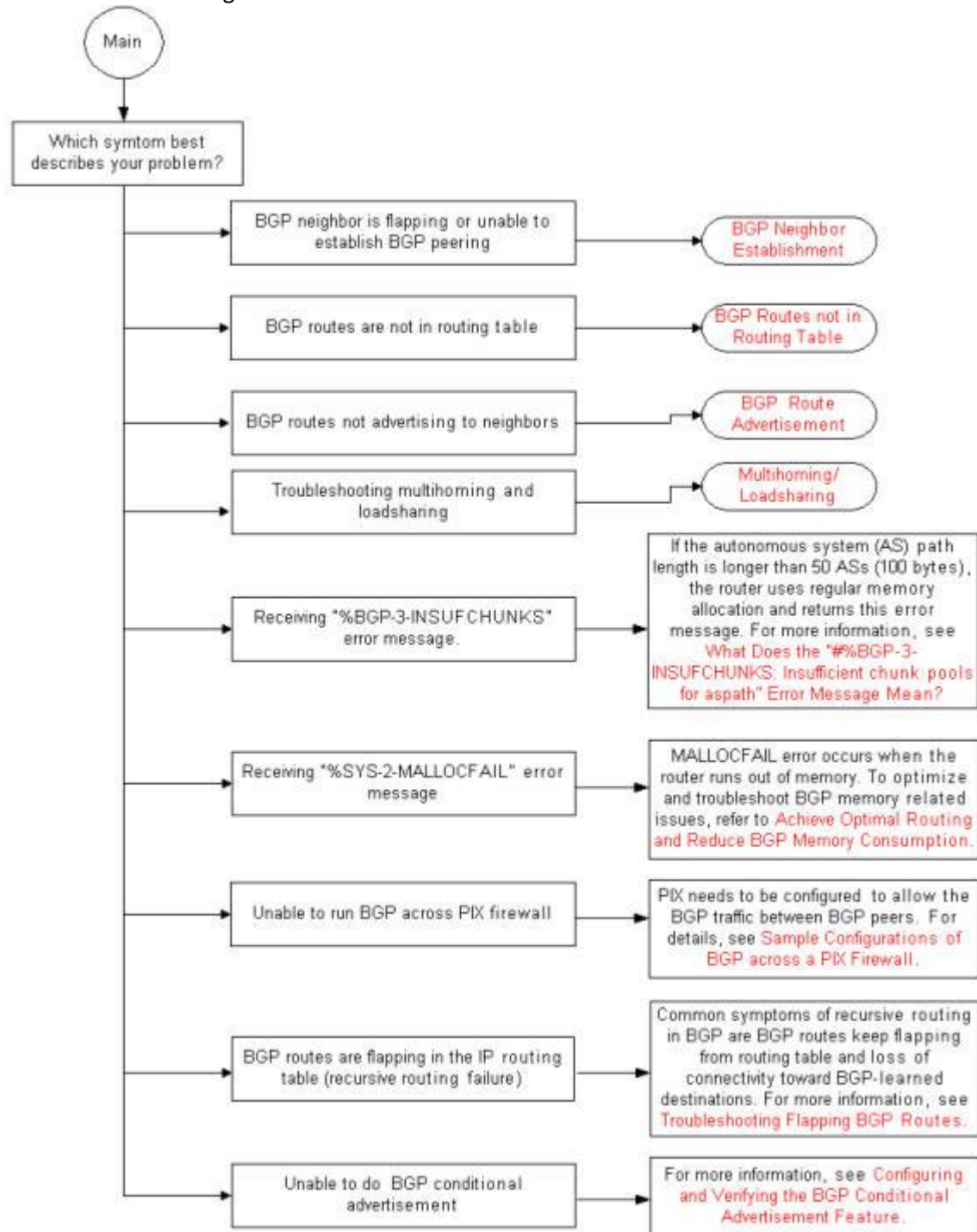


Abbildung 22: BGP Troubleshooting-Prozess



1.4 Deviceanalyse

Ein Netzwerk besteht in den meisten Fällen aus einer heterogenen Mischung von Geräten. Diese unterscheiden sich sowohl in Funktionalität, ihren Rollen und die von Ihnen unterstützten Protokollen und Software.

1.4.1 Geräte in einem Netzwerk

Router

Ist ein Netzwerkgerät, welches Netzwerkpakete zwischen verschiedenen Netzen weiterleiten kann. Die häufigsten Anwendungsfälle sind die Internetanbindung, die Kopplung mehrere Standorte oder direkt mehrere lokale Netzwerksegmente zu koppeln

Aus Sicht der Netzwerkschichten arbeiten sie auf der Schicht drei, sie treffen Entscheidungen über die Weiterleitung basierend auf IP Adressen. Eine weitere Funktion ist das Übersetzen zwischen privaten und öffentlichen IP Adressen, NAT/PAT, einige Router können auch Firewall Funktionen übernehmen.

Switch

Ein Switch ist ein Netzwerkgerät, welches verschiedene Geräte zu einem Netzwerk verbindet. Switches verwenden die Hardware-Adresse, auch MAC Adresse genannt, zur Adressierung der Pakete. Sie arbeiten auf dem Layer 2 des OSI-Modells.

Es gibt auch Switches, welche auf dem Layer 3 arbeiten können und somit Routing Funktionen übernehmen und IP Adressen verwenden. Solche Switches nennt man auch Layer 3 Switches.

Firewall

Eine Firewall ist ein Sicherungssystem welches ein Netzwerk vor unerwünschten Zugriffen schützt. Die Firewall Software beschränkt Netzzugriffe basierend auf Absender oder Ziel und dem genutzten Dienst. Entscheidungen welche Daten durchgelassen werden, entscheidet die Firewall basierend auf vordefinierten Regeln. Eine Firewall erkennt keine Angriffe, sondern setzt vordefinierte Regeln in der Netzwerkkommunikation um.

1.4.2 Probleme durch Geräte

Je nach Version des Geräts und der darauf installierten Software können sie nicht alle die gleichen Zusatzsysteme und Funktionen unterstützen. Eigenschaften, die Einfluss haben auf die Unterstützung von Funktionen sind Hersteller, Version des Gerätes und auch die Version des Betriebssystems auf dem Gerät.

Grundsätzlich kann man auf alle Geräte über eine Konsole zugreifen einige bieten auch Support für eine API an.



1.5 Erweiterungsanalyse

In diesem Abschnitt werden die während der Bachelorarbeit möglichen neuen Funktionalitäten für OATS aufgelistet und der Status-Quo analysiert.

1.5.1 Event-Korrelation

Events können über Beziehungen untereinander verfügen und somit ist es möglich, dass mehrere Problemmeldungen eingehen, die alle auf dasselbe Problem hindeuten. Durch Verwendung einer oder mehreren Open Source Lösungen sollte es möglich sein, solche Korrelationen zu entdecken, die korrekte Schlussfolgerung daraus zu ziehen und schliesslich das vorhandene Problem zu beheben.

Stand SA

In der SA wurde ein Client entwickelt, der das Komprimieren von Events ermöglicht (wurde fälschlicherweise als Aggregator bezeichnet, wie sich nach der BA-Recherche rausstellte).

So wurde als Reaktion auf eine bestimmte Syslog-Meldung ein Timer gestartet, welcher für eine bestimmte Zeit die Events der gleichen Art zählt und nach Ablauf der Zeit ein neues Event generiert, abhängig von der Anzahl der gezählten Events.

Diese Korrelation funktioniert zurzeit nur für ein bestimmtes Event. Es soll aber möglich sein eine vielfältige Art von Events zu korrelieren, ohne dass der Client umprogrammiert werden muss. Dies ist auch Gegensand der BA.

Definition

Ein **Event** bezeichnet im IT-Kontext eine Nachricht über etwas, was passiert ist und enthält Details wie Zeitpunkt und Ort des Events. So sind z.B. der Mausklick eines Users und das Runterfahren eines Routers, Events.

Event-Korrelation bedient sich solcher Events und analysiert diese um Beziehungen innerhalb der Events zu erkennen. Aus diesen Beziehungen lässt sich der Ursprung von verschiedenen Problemen ableiten und ermöglicht es zeitig eine Problemlösungsroutine auszulösen.

Die Daten für die Event-Korrelation können in Echtzeit gesammelt werden, aber es ist auch denkbar, dass man Daten analysiert, die über einen längeren Zeitraum gesammelt und gespeichert wurden. Im Kontext des automatischen Troubleshootings ist es nötig, dass die Daten in Echtzeit analysiert werden. Nichtsdestotrotz können auch in dem Fall wichtige Schlüsse aus Langzeitdaten gezogen werden und dürfen nicht ausser Acht gelassen werden.

Ein **Fehler** im System ist das auslösende Event für ein automatisches Troubleshooting, also der Ursprung auf den man mittels Event-Korrelation stossen möchte.

Event-Korrelations-Operationen

In diesem Abschnitt werden einige möglich Operationen vorgestellt, welche die Grundbausteine für die Event-Korrelation bilden [5].

Aus diesen Operationen lässt sich bereits eine Korrelations-Engine zusammenstellen. Dabei ist aber der Ablauf immer fix vorgegeben und man muss bereits über gewisse Muster unter den verschiedenen Events Bescheid wissen.

Kompression

Bei der Kompression von Events werden mehrere Events der gleichen Art durch ein einzelnes Event ersetzt. Das neue Event sollte Informationen darüber enthalten, welche Art von Event es ersetzt und wie viele Events es komprimiert.

Logische Operationen



Hierbei handelt es sich um Operationen der booleschen Logik, wie z.B. die UND-Verknüpfung zweier Events.

Aggregation

Aggregation bezeichnet das Ersetzen von mehreren nicht zwingend gleichartiger Events durch ein einzelnes Event.

Filtering

Beim Filtering handelt es sich um das Entfernen von nicht gewünschten Events aus dem Event-Stream.

Thresholding

Thresholding (dt. Schwellenwertbildung) bezeichnet das Erzeugen von Events, wenn ein festgelegter Schwellenwert überschritten/unterschritten wird (z.B. Antwortzeit < 100ms).

Rate Limiting

Dabei wird nur eine bestimmte Anzahl eines Events über eine gewisse Zeit weitergeleitet. So werden z.B. in 10 Sekunden nicht mehr als 10 Events der gleichen Art weitergeleitet.

Zeitliche Beziehungen

Zeit spielt bei der Event-Korrelation eine wichtige Rolle. So müssen für gewisse Problemfälle nur die aufgetretenen Events der letzten 20 Sekunden ausgewertet werden, während andere Problemfälle nur erkannt werden können, wenn auch die Daten der letzten Woche verfügbar sind. Deswegen ist entscheidend, dass man auf die zeitlichen Beziehungen unter den Events achtet.

Clustering

Clustering bezeichnet das Kombinieren von mehreren der vorher genannten Operationen.

Ansätze zur Event-Korrelation

Der grundsätzliche Ablauf einer Event-Korrelation ist beim gegebenen OATS schon gut vorstellbar. So werden von den Sensoren (Syslog, Streaming-Telemetry) Events generiert, welche in eine Pipeline eingespielen werden. Die Events in der Pipeline werden schliesslich von einem Client verarbeitet. Bei der Verarbeitung (bzw. Event-Korrelation) sind mehrere unterschiedliche Ansätze denkbar, die in Kurzform in diesem Kapitel erläutert werden. Für eine genauere Spezifikation der Korrelationstechniken und weitere Techniken wird auf [5] verwiesen.

Regel-basierte Korrelation

Dabei werden direkt im Code oder auch in einer externen Datei feste Abläufe festgelegt, was bei einem Event für Operationen und Aktionen anzuwenden sind. Ein solcher Regelsatz könnte so aussehen: «Wenn *Event A* 10-mal während 5 Minuten auftritt ist *Event B* eingetreten. Um das durch *Event B* eingetretene Problem zu lösen muss *Massnahme C* ausgeführt werden».

Vorteile:

- Der grösste Vorteil dieses Ansatzes ist die überschaubare Komplexität. Es ist möglich schnell einen Fall umzusetzen, der sich von Anfang bis Ende sauber durchziehen lässt.
- Es ist leicht nachvollziehbar, was in einem gegebenen Regelsatz passiert.
- Gut erweiterbar. Es ist einfach das entwickelte System um neue Regelsätze zu erweitern.

Nachteile:

- Ein implementierter Regelsatz ist in der Lage jeweils nur einen einzigen Problemfall zu behandeln.



- Neu auftretende Problemfälle müssen erst von jemanden erkannt und manuell implementiert werden.
- Es ist nicht möglich dynamisch auf Änderungen in der Domain einzugehen. Ändert sich etwas, sodass sich auch die Behandlung eines Problems ändert, so muss der Regelsatz manuell angepasst werden.

Fall-basierte Korrelation

Bei der Fall-basierten Korrelation wird jedes Problem und die dazugehörige Lösung als ein zusammenhängender Fall behandelt. Um das Problem zu lösen wird in einer Datenbank überprüft ob in der Vergangenheit schon ein ähnlicher Fall eingetreten ist und es wird versucht den aktuellen Fall ähnlich zu lösen.

Vorteile:

- Menschen behandeln Probleme auf die gleiche Art, ist somit intuitiv verständlich (learn → adapt → act).
- Eine stetig wachsende Problemfall-Datenbank deckt ein immer grösseres Spektrum von Problemen ab.
- Es können evtl. auch unbekannte Probleme mit bereits bekannten Lösungsansätzen behoben werden.

Nachteile:

- Das Klassifizieren von «ähnlichen» Problemfällen ist schwierig und ist meistens nur durch menschliche Intuition möglich, ist also schwierig in Code abzubilden.
- Dies gilt auch für das Anpassen von bereits bestehenden Lösungen.
- Allgemein ist das Verfahren sehr reaktiv und schwierig in Code abzubilden, also nicht ideal für automatisches Troubleshooting.

Modellbasierte Korrelation

Die Idee der modellbasierten Korrelation ist es, die Struktur und das Verhalten des unter Beobachtung stehenden Systems in einem Modell abzubilden. Sind Verhalten und Struktur des gesamten Systems bekannt, ist es relativ einfach den Ursprung eines Problems durch logische Argumentation zu erkennen.

Dieser Ansatz ist nicht gut für Netzwerke geeignet, da die Beschreibung des gesamten Systems und dessen Verhalten sehr aufwändig ist (physikalische Architektur vs. logische Architektur).

Modellbasierte Korrelation ist besser geeignet für Systeme, in denen das System bereits in der Planungsphase genau spezifiziert wird (z.B. elektrische Stromkreisläufe).

Fehlerlokalisierung durch Abstimmung

Hierbei muss jeder Knoten in einem System seine Meinung zu einem aktuellen Problemfall abgeben. Der Ursprung des Problems wird somit durch die Mehrheit der Stimmen bestimmt.

Dieser Ansatz ist für ein automatisches Troubleshooting zu ungenau, da er nur auf den Ort des Problems hinweisen kann. Er macht z.B. keine Angaben darüber, welcher Prozess oder welches Protokoll für den Fehler verantwortlich ist. Zudem wird muss jeder Knoten im Netzwerk bekannt sein.

Es ist aber durchaus denkbar, dass ein solcher Ansatz in einzelnen Problemfällen als Zwischenschritt hilfreich sein kann.

Explizite Fehlerlokalisierung

Bei diesem Ansatz wird vorgeschlagen, jedem Event sämtliche Informationen beizufügen. Dann werden alle Events die in einem bestimmten Zeitraum eingetroffen sind zusammen analysiert und der Punkt in dem sich alle Events überschneiden ist verantwortlich für den Fehler.

In einem Netzwerk setzt dies voraus, dass einem Problemfall nur ein einzelnes Gerät zugrunde liegt, obwohl auch mehrere Geräte verantwortlich sein könnten. Zudem ist es schwierig, sämtliche



benötigte Informationen zu jedem Event anzufügen (einige Informationen müssen oft gesammelt werden, nachdem ein Event eingetroffen ist).

Dependency-Graph

Ein Dependency-Graph stellt Abhängigkeiten zwischen Knoten in einem System dar. In einem Netzwerk stellen diese Knoten Netzwerkgeräte dar und die Abhängigkeiten sind Routing-Protokolle wie z.B. OSPF.

Bei einem Event wird auf dem Dependency-Graph überprüft, wo der Ursprung eines Fehlers liegen könnte. Dies wird in OATS in kleinem Rahmen bereits durchgeführt, indem bei einem OSPF-Problem die Nachbarn eines Event-Ursprungs überprüft werden.

Ein komplettes Troubleshooting ist nur durch den Dependency-Graphen kaum möglich, kann aber ein sinnvoller Zwischenschritt für einzelne Problemfälle sein.

Machine Learning

Machine Learning bezeichnet das Konstruieren und Anwenden von Algorithmen, welche mittels Daten trainiert werden und versuchen Vorhersagen zu treffen. Um diese Vorhersagen zu treffen bedienen sich die Algorithmen der statistischen Mathematik und der Wahrscheinlichkeitstheorie. Gerade im Kontext von automatischem Troubleshooting wäre dies ein sehr mächtiges Konzept. Auch dieser Ansatz hat wieder Vor- und Nachteile:

Vorteile:

- Erkennt auch sehr komplexe Korrelationen, welche von Menschen nicht erkannt werden können.
- Erlaubt somit ein vollständigeres Abdecken der auftretenden Probleme in einer gegebenen Domain.

Nachteile:

- Setzt vertiefte mathematische Kenntnisse voraus. Es muss genau analysiert werden, welcher Machine Learning Algorithmus angewendet werden soll. Wird der falsche Algorithmus gewählt, sind die Vorhersagen die die Maschine trifft wahrscheinlich nutzlos.
- Benötigt eine grosse Menge an Daten. Die Vorhersagen werden besser, wenn mehr Daten verfügbar sind.
- Benötigt eine hohe Datenqualität. Die Daten die der Maschine zugeführt werden, müssen unabhängig sein und einen hohen Informationsgehalt haben. Dies setzt hohe Ansprüche an die Datensammlung und Datenaufbereitung.
- Es ist oft schwierig nachzuvollziehen, wie der Machine Learning Algorithmus zu seinen Ergebnissen kommt. So muss der Anwender hoffen, dass er alles richtig gemacht hat und den Ergebnissen vertrauen.

Umsetzung

Die vielversprechendsten Methodiken die in OATS eingesetzt werden könnten, sind die Regelbasierte Korrelation und Machine Learning.

Die anderen Ansätze verfügen jeweils über einen oder mehrere Aspekte, die schwierig mit Programmiersprachen abzubilden sind.

So ist es bei der Fall-basierten Korrelation schwierig zu definieren was Ähnlichkeit bedeutet. Für die modellbasierte Korrelation müsste das gesamte System erfasst werden, was in der Realität kaum machbar ist. Eine Fehlerlokalisierung durch Abstimmung ist nur möglich, wenn die einzelnen Knoten eine «Meinung» abgeben können, was bei Netzwerkgeräten schwierig ist. Explizite Fehlerlokalisierung ist ein relativ naiver Ansatz, so kann man nicht immer davon ausgehen, dass einem Problem nur ein Fehler zugrunde liegt.

Dependency-Graphen werden beim bestehenden OATS-System bereits indirekt eingesetzt. So sind in der Datenbank erfasst, welche Nachbarn eines Netzwerkgeräts über welches Netzwerkinterface erreichbar sind.



Für die Bachelorarbeit würde es am meisten Sinn machen, mit einer Regel-basierten Korrelation zu beginnen, da sich damit gut und schnell testen lässt, ob die Architektur des Systems so funktioniert wie gewünscht.

Ist eine gut funktionierende Systemarchitektur vorhanden, lässt sich im Nachhinein immer noch ein Client für Machine-Learning hinzufügen. Allgemein ist dies ein guter Ansatz, da Machine Learning erst Sinn macht, wenn bereits möglichst viele strukturierte Daten vorhanden sind.

1.5.2 Datenbank

Stand SA

Zurzeit verwenden wir für alle Daten die ausserhalb von Salt gespeichert werden eine MongoDB. Da OATS nun erweitert und ausgearbeitet wird, kommen weitere Daten und Informationen dazu, die ebenfalls in die Datenbank gespeichert werden.

In der Datenbank arbeiten wir mit drei Unterkategorien, bei MongoDB Kollektionen genannt, die jeweils unterschiedliche Arten von Datensätzen abspeichern. Diese sind Netzwerkgeräte, Techniker und die bearbeiteten Problemfälle.

Die kleinste davon ist die Techniker Kollektion, hier wurden jeweils Namen und ein Slack Channel gespeichert auf welchem man sie erreichen kann. Die Techniker haben ausserdem eine Relation zu den Problemfällen, um zu markieren welche sie bearbeitet haben oder gerade bearbeiten. Man hat hier also keine komplexen Daten und nur einfache Relationen, daher sind diese Daten hier in einem guten Zustand.

Evaluation der bestehenden Lösung

Aufsetzen der Netzwerk Daten

Um OATS zu deployen müssen diverse Schritte unternommen werden, bis das System lauffähig ist. Einer der umständlichsten Schritte ist dabei das Aufsetzen der Netzwerk Datenbank. Hier muss zuerst die Datenbank aufgesetzt werden und danach die Daten über das Netzwerk in diese zu importieren. Es kann aber sein, dass diese Daten nicht sauber vorhanden sind oder bei einem neuen Netzwerk erst noch erfasst werden müssen. Es wäre gut, wenn dieser Schritt etwas weniger umständlich wäre, da er meist jedoch nur einmal durchgeführt wurde ist er nicht eine Automatisierungs-Priorität, wichtiger wäre hier eine saubere und übersichtliche Darstellung der bereits erfassten Daten, um zu sehen wo es noch Lücken hat und welche man bereits eingetragen hat.

Netzwerk Daten

Zu den Netzwerk-Daten gehören die einzelnen Netzwerk Geräte und ihre Verbindungen untereinander. Zurzeit werden die Geräte einzeln abgespeichert und als Untertabelle jeweils die Verbindungen zu den benachbarten Geräten. Diese Netzwerk Daten sollen als SoT (Source of Truth) fungieren, in dem sie den idealen Zustand des Netzwerks aufzeigen.

Zurzeit benötigen wir die folgenden Funktionen für unseren Troubleshooting-Prozess:

- `get_ospf_neighbors`
- `get_vrf_ip`
- `get_interface_neighbor`
- `get_neighbors`

Wir möchten hier hauptsächlich wissen welche Nachbarn ein Gerät hat und wie man diese erreicht. Also spielen die Relationen zwischen den einzelnen in der Datenbank eingetragenen Geräten eine wichtige Rolle. Hier stösst die NoSQL Technologie der MongoDB an seine Grenzen, da kein Schema erzwungen wird. Innerhalb einer Datenbank kann es also sein das hier Relationen fehlerhaft eingetragen werden. Die Anforderung an unsere Datenbank, dass sie Relationen und Daten konsistent und korrekt hält ist somit nicht erfüllt.



Sensor Daten

Da für eine effiziente Event-Korrelation die frischen Daten in Echtzeit mit alten Daten verglichen werden müssen, ist es nötig ein eigenes Datenbankkonzept für diese Daten zu erstellen.

Da alle gelieferten Daten zeitabhängig sind, bietet sich hier eine sogenannte «time series database (TSDB)» an. Diese Art von Datenbank ist darauf optimiert solche Timeseries-Daten, welche aufzeigen wie sich ein System im Verlaufe der Zeit verändert, zu verarbeiten.

Typische Eigenschaften von solchen TSDBs sind:

- Schnelle Abfragezeiten (<100ms)
- Effizientes Speichern von grossen Datenvolumen (ermöglicht einfache Skalierung)
- Komprimierung von alten Daten (je älter Daten werden, desto weniger relevant werden sie)

Beispiele von TSDBs sind InfluxDB [6] oder OpenTSDB [37]. Bei Verwendungen und Verarbeitung von Streaming Telemetrie Daten ist man hier stark daran gebunden eine solche spezifische Art von Datenbank zu wählen, da andere Lösungen die Anforderungen an die Performance nicht erfüllen.

Reports

Die Reports zu den Problemfällen ist die letzte der drei Kategorien. Diese werden angelegt sobald ein Fall erstellt wurde, während der Bearbeitung der Reports durch das System werden daran laufend Updates abgespeichert und schliesslich werden sie als abgeschlossen markiert. In OATS verwenden wir folgende Zugriffe auf die Datenbank im Rahmen der Bearbeitung eines Problemfalls:

- Create_case
- Update_case
- Take_case
- Close_case

Die Cases Daten haben hier jeweils eine Relation zu den Technikern falls dieser gerade daran ist einen Case zu bearbeiten und zu den Netzwerkgeräten zur zu Ordnung von welchem Gerät aus der Troubleshooting Fall gestartet wurde.

Eine Überlegung wäre hier, nur die Laufenden Reports in der Datenbank zu halten und solche, welche durch einen Techniker noch bearbeitet werden müssen, um ein Problem zu beheben. Für komplett abgeschlossene Fälle macht es nicht viel Sinn diese in der Datenbank zu halten, da diese mit der Zeit wachsen werden und man damit die Übersicht verliert. Sinnvoll wäre hier das Halten einer Statistik über die Informationen in den Fällen, wie etwa welche Geräte ein Problem hatten oder welche Protokolle involviert waren. Die genaue Information mit den Behebungsschritten müsste nicht erhalten bleiben, vor allem bei automatisch bearbeiteten und erfolgreich gelösten Problemen ist dies redundant.

Mögliche Lösungen

Wenn wir nun unsere Anforderungen an die Datenbank betrachten kommen wir auf folgende Hauptpunkte:

- Open Source
- Beliebige Netzwerkgeräte
- Konsistente Relationen
- Einfaches Ansteuern mit OATS
- Zusätzliche Features

Wir möchten uns nun überlegen, ob wir diese Anforderungen erfüllen und welche wir Möglichkeiten haben.

In einem kommerziellen Umfeld gibt es oft Vorgaben vom Unternehmen aus, welche Datenbank man verwendet, diese Entscheidung trägt sich dann auch auf den Netzwerkbereich über, man kann also nicht immer die ideale Variante wählen. Für uns ist hierfür also wichtig, dass es relativ unkompliziert



ist die Datenbank zu wechseln. In OATS benutzten wir für den Datenbankzugriff das OATSDBhelpers-package, welche von den Troubleshooting-Fällen aufgerufen werden und dann die Datenbank ansteuern.

Graphdatenbanken vs. relationale Datenbanken

Wenn wir nur Datenbanken betrachten gibt es zwei sinnvolle Möglichkeiten, die sicherstellen, dass die Relationen zwischen den einzelnen Geräten konsistent sind. Diese sind relationale und Graphdatenbanken. Die Graphdatenbanken werden verwendet, um stark vernetzte Informationen darzustellen und abzuspeichern. Hier könnten wir die Netzwerkgeräte als Knoten abspeichern und die jeweiligen Verbindungen untereinander als Kanten. Somit könnten wir die Konsistenz der Relationen untereinander sicherstellen. Auch hier gibt es Open Source Lösungen, welche wir verwenden könnten um diese Anforderung zu erfüllen. Eine davon wäre neo4j [38]. Das Problem hier wäre das Aufsetzen der Datenbank, da hier Relationen sehr umständlich erstellt werden müssen, ausserdem sind die Anbindungen an externe Tools oft umständlich, da komplizierte Join-Operationen durchgeführt werden müssen. Änderungen am Schema oder an einzelnen Objekten würden zudem zusätzliche Arbeit benötigen. Zudem gibt es hier wenige weitere Features, welche für uns interessant sind. Obwohl die Graphdatenbank Vorteile hat sind die Nachteile überwiegend und wir werden sie nicht genauer analysieren.

Für eine genaue Entscheidung möchten wir uns nun zwei Lösungen, die im Netzwerkbereich bereits häufig eingesetzt werden, genauer ansehen. Diese Lösungen sind Netbox und APIC-EM.

Netbox

Umfassende Lösungen wie IP-Address-Management (IPAM) oder Data Center Infrastructure Management Software (DCIM) bieten oft graphische Oberflächen und zusätzliche Funktionen an, die für den Gebrauch im Netzwerkbereich interessant sind. Sie verwenden im Hintergrund eine normale Datenbank. Man kann hier entweder die API der Software ansteuern oder direkt die darunterliegende Datenbank. Eine Variante einer IPAM ist Netbox [14], dies ist eine Open Source Web-Applikation die im Hintergrund eine PostgreSQL Datenbank verwendet und eine Visualisierung des Netzwerks bietet. Mit solch einer Lösung könnte man gleich zwei Erweiterungen und Optimierungen ermöglichen. Zum einen würden wir mit der Verwendung der PostgreSQL Datenbank die Anforderungen an die relationale Konsistenz der Datenbank abdecken. Zum Zweiten hätten wir hier direkt auch eine Visualisierung der gespeicherten Daten und es erlaubt es uns einen einfacheren Zugriff auf diese Daten, welche oft unübersichtlich sein können.

Features

Eine Visualisierung der Daten über eine Web-Applikation ist das wichtigste Feature von Netbox. Das Dashboard, wie unten ersichtlich, erlaubt es einem schnell zu überprüfen ob alle Daten vorhanden sind und ob diese korrekt sind. Dies erleichtert vor allem das Eintragen der Daten beim Aufsetzen. Für das Aufsetzen kann man zusätzlich einen Daten-Import oder -Export durchführen, um etwa ein Backup durchzuführen oder einen neue Netbox aufzusetzen. Netbox selbst bietet kein eigenes Monitoring oder Konfigurations-Management, sondern dient dazu, externen Tools Daten zu liefern.



Abbildung 23: Netbox-UI

Mit der im Hintergrund liegenden PostgreSQL Datenbank und dem Schema, geliefert von Netbox, können wir sicherstellen, dass unsere Daten, vor allem die Verbindungen zwischen einzelnen Geräten, konsistent sind. Weiter besitzt PostgreSQL einige netzwerkrelevante Datentypen, wie etwa «inet», «cidr» oder «macaddr», was hilfreich ist, wenn es darum geht korrekte Daten einzulesen.

Einbindung an OATS

Wie bereits oben erwähnt gibt es zwei Möglichkeiten wie man von Netbox Daten in unser System einspeisen könnte. Das direkte Ansprechen von Netbox über die angebotene API oder die Kommunikation mit der unterliegenden Datenbank. Die angebotene API ist sehr umfangreich und bietet uns eine einfache Möglichkeit an verschachtelte Daten heranzukommen, wie etwa die Nachbarn, welche hinter einem Interface eines Geräts liegen. Wir können also unsere vorhandenen OATSDBhelpers auf die neue API umschreiben, so dass wir diese Aufrufe nicht direkt aus den Clients heraus erledigen.

Wenn wir uns für Netbox als unsere Datenbank entscheiden würden, müssten wir zudem die Problemfälle-Daten in die neue Datenbank migrieren, da es wenig Sinn ergeben würde hier zweispurig zu fahren. Dazu müssten die betroffenen Funktionen, wie etwa das Erstellen eines neuen Problemfalls in der Datenbank, komplett umgeschrieben werden, um diese in die neue Datenbank zu schreiben. Da wir jedoch den Datenbankzugriff aus dem Troubleshooting Skript entkoppelt und in ein eigenes Package verpackt haben, sollte dies kein Problem darstellen.

APIC-EM

Eine weitere Möglichkeit wäre APIC-EM [39] von Cisco, was für Application Policy Infrastructure Controller Enterprise Module steht. Hierbei handelt sich um eine Plattform, welche verschiedene Features anbietet, welche man über eine REST-API erreichen kann. Besonders interessant sind hier für uns die Network Information Database (NIDB) und die Network Topology Visualization. Während eine reine Datenbank, wie wir sie bis jetzt verwendeten, nur einen Datenspeicher bietet, so bietet Netbox etwa bereits eine Visualisierung dieser Daten. Der APIC hingegen bietet noch weitere Features an, die weit über eine einfache Visualisierung der Daten hinausgehen.



Features

Die bereits erwähnte Network Information Database hat das zusätzliche Feature, dass man das ganze Netzwerk scannen kann, um die aktuellen Daten zu erhalten. Neue Netzwerkgeräte die angeschlossen werden, können so direkt entdeckt werden. Die graphische Ansicht der Netzwerk Topologie wird ebenfalls automatisch erstellt.

Mit dem Network Plug-and-Play bietet der APIC die Funktion, neu angeschlossene Geräte direkt mit dem richtigen OS und der richtigen Konfiguration zu versorgen. Dies ist vor allem für den Rollout von neuen Geräten interessant, für uns also nicht besonders interessant, da wir uns mehr auf ein bereits existierendes Netzwerk konzentrieren.

Die Path-Trace Funktion hingegen kommt unseren Interessen näher, hier handelt es sich um ein Feature, welches uns erlaubt anhand Quell- und Zieladresse den Pfad von Paketen innerhalb des Netzwerks anzuzeigen.

Es gibt zudem eine separat lizenzierte Cisco Intelligent WAN (IWAN) Applikation welches das Aufsetzen eines kompletten WAN Systems erlaubt. Hier werden Netzwerk Profile automatisch aus dem System generiert, nachdem man sie im IWAN Design zusammengestellt hat.

Ein weiteres separat lizenziertes Produkt ist die Enterprise Service Automation (ESA) Applikation, mit diesem kann man automatisch Netzwerk Services deployen, verwalten und beobachten. Es erlaubt einem die optimale Performance im Netzwerk sicherzustellen.

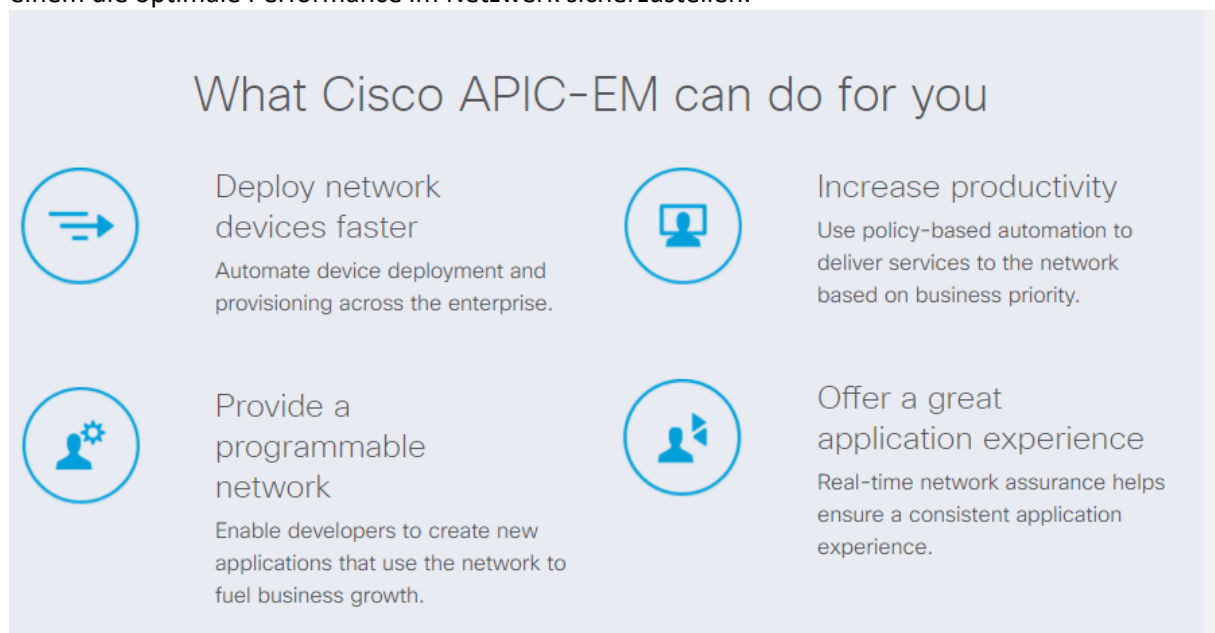


Abbildung 24: APIC-EM Features

Einbindung an OATS

Wie erhalten wir also Daten aus dem Cisco APIC-EM in OATS. Dazu liefert sie eine eigene REST-basierte API, welche man mit HTTP ansprechen kann. Wir könnten also auch hier das OATSDBhelpers Package so umschreiben, dass es die API anspricht.

Die Problemfälle welche wir ebenfalls in der Datenbank speichern, könnten wir in unserer Datenbank lassen, da der APIC zwar die Netzwerkgeräte in eine eigene Datenbank speichert aber man nicht noch eigene Daten dort abspeichern kann.

Die APIC-EM bringt aber einige Nachteile mit sich. So ist es eine Cisco Proprietäre Software und Hardware und nicht eine Open Source Software, wie wir es bis jetzt benutzt haben und jeweils auch



diese Anforderung an alle evaluierten Tools gestellt haben. Weiter ist die Verwendung des APICs mit einigen Kosten verbunden, grundsätzlich gibt es eine Freie Lizenz mit allen Grundfunktionen, um jedoch die volle Kapazität auszunutzen benötigt man eine Lizenz, ein genauer Kostenbetrag publiziert Cisco nicht, sondern man müsste sie mit den genauen Anforderungen kontaktieren. Der zweite Kostenpunkt kommt in Form von Vendor-Locking, so bietet der APIC-EM seine Funktionen nur für Cisco Geräte. Dies stellt aber für uns ein Problem dar, da wir uns nicht nur auf einen Gerätehersteller begrenzen möchten. Ein weiterer Kostenpunkt ist die benötigte Leistung: für eine virtuelle Applikation auf einem Hypervisor oder direkt auf einem Server sieht die benötigte Leistung wie folgt aus:

- VMware ESXi Version: 5.1/5.5
- Server: 64-bit x86
- Virtual CPU (vCPU): 6
- CPU speed: 2.4 GHz
- RAM: 64 GB (Single Node), 32 GB (Per Host for Multi-Node)
- Storage: 500 GB of available or usable storage after hardware RAID
- RAID level: Hardware-based RAID at RAID level 10
- Disk I/O speed: 200 MBps
- Network adapter: 1 or more
- Browser: Chrome (44.0 or later)
- Web access required: Outbound secure web (HTTPS) access from the Cisco APIC-EM to the Internet for automatic updates of the controller software

Schlussfolgerung

Wir haben nun einige auf unseren Anforderungen basierte Lösungen betrachtet. Eine einfache Datenbank, ohne zusätzliche Visualisierung oder Erleichterung der Bedienung, reicht für uns nicht aus. Weshalb wir definitiv unsere Daten migrieren müssen. Die weiter evaluierten Lösungen Netbox und APIC-EM bieten beide mehr als bloße Datenbanken.

Entscheidungskriterium	Cisco APIC-EM	Netbox
Open Source	Nein	Ja
Unterstützte Plattformen	Cisco	Beliebig
Konsistente Relationen	Ja	Ja
Einfaches ansteuern mit OATS	Ja, durch API	Ja, durch API
Zusätzliche Features	Visualisierung Plug-and-Play Path-Trace IWAN ESA Import/Export Backup	Visualisierung Import/Export Backup
Speichern der Problemfälle	Separat	In Datenbank möglich aber ohne Support

Der Cisco APIC-EM bietet uns neben den Features welche uns auch von Netbox geboten werden noch einige weitere sehr interessante Features an. Jedoch ist das Weggehen von Open-Source Lösungen und das Vendor-Locking der unterstützten Netzwerk Geräte ein zu hoher Preis.

Wir haben uns deshalb entschieden in der Bachelorarbeit mit Netbox zu arbeiten.



1.5.3 Codebasis

Momentan ist Programmierwissen vorausgesetzt, um OATS zu erweitern. Es soll aber möglich sein, einfachere Problemfälle in OATS zu implementieren, ohne dass man Programmieren kann. Zu diesem Zweck soll ein Konzept entworfen und entwickelt werden, mit welchem z.B. über ein externes File neue Troubleshooting-Fälle in OATS eingefügt werden können. In den folgenden Kapiteln wird aufgezeigt, wie OATS zum Ende der Studienarbeit konfiguriert werden musste. Zusätzlich wird bewertet, inwiefern die Konfiguration Programmierwissen voraussetzt.

Grundsätzliche Konfiguration

Bevor man beginnt mit OATS zu arbeiten müssen folgende Konfigurationen vorgenommen werden:

- Napalm-logs: Listener und Publisher Adresse konfigurieren
- Salt Proxies [40]
 - Pillar für device
 - Referenz in top.sls
- Netzwerkgeräte konfigurieren
 - Privilegien
 - Syslog
 - Hostnames
 - Workflow-relevante Protokolle wie z.B. OSPF
- Eintragen der Geräte in die DB per Terminal

Die Napalm-logs Konfiguration ist ein einfaches Eintragen von IP Adressen und Port Nummer und basiert, wie auch die Salt-Konfiguration, auf YAML-Dateien.

Schliesslich müssen neu erfasste Geräte auch noch in die Datenbank eingetragen werden, hierzu haben wir ein Kommandozeilen Skript geschrieben, womit auch dies kein Problem sein sollte.

Die Konfigurationen benötigen also kein Programmierwissen. Teils wird vorausgesetzt, dass man mit der Kommandozeile vertraut ist oder das Schreiben eines YAML Eintrags. Hier sehen wir deshalb keinen Grund eine Änderung in der bestehenden Arbeitsweise vorzunehmen.

Neuer Workflow erfassen

Einen neuen Workflow zu erfassen ist mit den folgenden Schritten möglich, auch diese wurden im Detail in der Studienarbeitsanleitung erklärt weshalb sie hier nur aufgelistet und nicht weiter beschrieben sind.

1. Syslog-Meldung konfigurieren
2. Napalm-logs konfigurieren
3. Oatsclient/Listener ergänzen
 - Empfang der Daten von napalm-logs
 - Erstellen und Senden des neuen Events an den Salt-event-bus (mittels salt_event.py)
 - Aggregation von Events (mittels aggregate.py, nur falls nötig)
4. Salt-master benötigt Reactor Eintrag für den neuen Event Tag
5. Salt-reactor, Referenz auf das Lösungsfile (z.B. tshoot.ifdown) und angeben der relevanten Datenfelder
6. Salt-runner, Tshoot-Script
 - Startet und eröffnet einen neuen Fall in der Datenbank
 - Interpretiert die Informationen in der Yang-Meldung
 - Sammelt zusätzliche Informationen
 - Sendet Befehle zur Lösung an das Netzwerkgerät
 - Überprüft ob es nun gelöst wurde
 - Falls alles ok Fall abschliessen, sonst Meldung an Slack



Auch hier haben wir diverse Einstellungen die problemlos ohne Programmierwissen getätigt werden können. Wo zwingend verlangt wird, dass man programmieren kann ist das Aggregieren von Events im Oatsclient und schliesslich einen Grossteil des Tshoot Runner Skripts. Das Problem hier ist, dass diese jeweils sehr spezifisch auf die einzelnen Problemfälle sind und sich stark unterscheiden können.

Die sich im Tshoot befindende Logik wird wohl nicht abstrahiert werden können. Was wir aber hier könnten ist das Definieren von oft genützten Standard Funktionen zur Informationssammlung, wie wir es etwa für den Datenbankzugriff oder für Funktionen wie Ping haben. Auch könnte man den Zugriff auf Netzwerkgeräte weiter abstrahieren, so dass man in einem Yaml File schreibt auf welches Gerät man zugreifen möchte, die gewünschten Änderungen hingegen benötigen Wissen über das Netzwerkgerät.

Allgemein sind bereits viele der Konfigurationen und nötigen Einstellungen rund um den Workflow problemlos ohne Programmierwissen möglich. Das Schreiben einer Workflow Logik ist zu komplex um diese auf Standardfunktionen und Optionen zu reduzieren, jedoch könnte man es vereinfachen in dem man gewisse Funktionen bereitstellt, wie etwa ein Zugriff auf Geräteinformationen und schliesslich daraus ein Gerüst generiert mit dem man weiterarbeiten kann.

Am meisten Potenzial zur Vereinfachung des Erweiterns des Systems bietet wohl der napalm-logs-client.

1.5.4 Sensoren

Stand SA

OATS ist zurzeit in der Lage Syslog-Messages zu verarbeiten. Dabei benutzt es das Tool napalm-logs, welches die plaintext Syslog-Messages entgegennimmt und sie in eine Form bringt, in welcher die Daten im Code verarbeitet werden können.

Für die Bachelorarbeit soll OATS um mindestens einen Sensor erweitert werden, damit das System nicht auf Syslog limitiert ist. Dafür kommen die in Kapitel 1.2 (Technologieanalyse) vorgestellten Protokolle/Technologien SNMP, Netflow, NETCONF, IP SLA und Streaming-Telemetry in Frage. SNMP wurde bereits in der SA ausgeschlossen, da es ein relativ alter und limitierter Standard ist.

Für die Bachelorarbeit werden somit noch IP-SLA, Netflow, NETCONF und Streaming-Telemetry als mögliche Sensoren in Betracht gezogen.

Netflow

Netflow dient zur Analyse des Netzwerkverkehrs auf einem Interface.

Es ist relativ einfach zu konfigurieren und es existiert bereits ein Use Case dafür (siehe Unterkapitel 1.5.7, Weitere Troubleshooting-Fälle), in welchem der Ursprung einer zu hohen Netzwerklast detektiert werden soll.

IP SLA

IP SLA kann in OATS auf zwei Arten verwendet werden:

1. **Passiver Sensor**
2. **Aktive Datensammlung**

Als passiver Sensor kann IP SLA konfiguriert werden, indem direkt auf dem Gerät Schwellenwerte für z.B. Jitter, Latenz oder Packet-Loss konfiguriert und überwacht werden. Überschreitet der aktuelle Wert den Schwellenwert, so wird eine SNMP-Trap mit den relevanten Werten ausgelöst. Dies stellt für OATS das Event dar, welches einen Troubleshooting-Prozess in Gang in setzt.



Bei der aktiven Datensammlung wird IP SLA nicht als Sensor eingesetzt, sondern sammelt als Teil eines Workflows Daten im Netz, um den Ursprung eines Fehlers zu bestimmen.

Das Hauptargumente, die gegen den Einsatz von IP SLA in OATS sprechen ist, dass IP SLA nur von Cisco Geräten komplett unterstützt wird. OATS hat das Ziel herstellerunabhängig zu sein und verzichtet deswegen wo möglich auf den Einsatz von Cisco proprietären Technologien und Protokollen.

Für den Einsatz von IP SLA sprechen die Funktionalitäten (Sammeln von Daten zu Jitter, Latenz, Packet-Loss etc.), sowie dass die IP SLA Daten über Streaming-Telemetry übertragen werden können (somit müssen keine SNMP-Traps eingesetzt werden). Diese Kombination ist sehr mächtig und überwiegt die möglichen Nachteile, vor allem weil die in der Testumgebung eingesetzte Streaming-Telemetry-Technologie auch Cisco-basiert wäre.

Deswegen ist es sinnvoll, dass wenn Streaming-Telemetry in das System eingebunden wird, auch die Funktionalitäten von IP SLA zu benutzen.

Streaming-Telemetry vs. NETCONF

Streaming-Telemetry und NETCONF wurden in früheren Kapiteln bereits analysiert. Es existiert aber noch kein expliziter Use Case für die beiden. In diesem Unterkapitel sollen die beiden direkt miteinander verglichen werden, um zu entscheiden, welches priorisiert wird.

Datenmodellierung

Der wichtigste Punkt bei beiden Technologien ist die Form in welcher die Daten geliefert werden. Diese ist bei beiden YANG-basiert und stellt somit kein Hindernis für eine Einbindung in OATS dar.

Datenübertragung

NETCONF benötigt eine aktive Session zwischen Router und Server und erlaubt nur eine begrenzte Anzahl von aktiven Sessions. Dies könnte zu Problemen bei der Skalierung des Netzwerks führen. Bei der Streaming-Telemetry wird auf dem Router eine IP und ein Port angegeben, an den die Daten geschickt werden sollen. Es wird keine Session aufrechterhalten und die Daten werden im Fire & Forget Stil abgeschickt. Im Kontext von OATS ist es verträglich, wenn einzelne Datenblöcke verloren gehen.

Datenkadenz

NETCONF funktioniert hier ähnlich wie Syslog, indem es eine Event-Notification an den NETCONF-Server schickt, sobald es auftritt. Es kann sein, dass dies nur sehr selten passiert, oder es können sehr viele Events in kurzer Zeit auftreten.

Streaming-Telemetry schickt die Daten in festgelegten Zeitabständen zum Server (z.B. alle 10 Sekunden). Es deckt hier also einen anderen Anwendungsfall als Syslog/NETCONF ab.

Konfiguration

Bei NETCONF muss als erstes eine Session zwischen dem Router und dem NETCONF-Server konfiguriert werden. Danach muss der Server noch angegeben, über welche Events er notifiziert werden will (per XML). Hier müssen auf beiden Geräten diverse Einstellungen geändert werden.

Bei der Streaming-Telemetry muss der Server lediglich IP+Port angegeben, auf welchem er auf die Daten hören will. Auf dem Router muss konfiguriert werden, wohin und wie er die Daten schicken soll und welche Daten er schicken soll. Dabei kann aus einer grossen Anzahl vordefinierter YANG-Modelle ausgewählt werden. Der Hauptaufwand ist hier auf dem Router aufzuwenden (per CLI/Remote-Konfigurationstool).

Zusätzlich bietet Streaming-Telemetry in der Theorie unbegrenzte Möglichkeiten. So erlauben es die bereits existierenden Modelle für IP SLA und Netflow, auch die Daten von anderen Sensoren zu



übertragen, was in der Theorie ein sehr mächtiges Konzept ist, da man somit nur ein Datenformat auf dem Server verarbeiten muss.

Netzwerklast

Da NETCONF über keine vordefinierte Datenkadenz verfügt, ist es möglich, dass in einer kurzen Zeit sehr viele Events generiert werden. Dies kann eine hohe Last für das Netzwerk zur Folge haben (wie bei Syslog auch).

Die Belastung für das Netz durch die Streaming-Telemetry ist stetig, da nur in regelmässigen Abständen Daten verschickt werden.

Fazit

Da NETCONF und Syslog eine ähnliche Funktionsweise haben, ist es interessanter, als Erstes Streaming-Telemetry ins System einzubinden. Auch verfügt die Streaming-Telemetry über einige Vorteile gegenüber von NETCONF, wie die vereinfachte Konfiguration, die reduzierte Netzwerklast und die Möglichkeit die Daten von anderen Sensoren zu streamen.

Anmerkung: Wenn im Netz IOS-XE Router eingesetzt werden, erübrigt sich diese Entscheidung, da Streaming-Telemetry bei IOS-XE über NETCONF-Sessions läuft.

1.5.5 Deployment Automatisierung

Zurzeit ist der Aufwand um OATS aufzusetzen relativ gross, da viele einzelne Tools (salt, napalm, napalm-logs etc.) mit ihren jeweiligen Dependencies aufgesetzt werden müssen.

Stand SA

Am Ende der Studienarbeit haben wir ein Anleitungsdokument aufgesetzt, wo wir die Schritte dokumentiert haben, welche man unternehmen muss um alle Software Komponenten zu installieren und zu konfigurieren, damit man schlussendlich OATS verwenden kann. Diese umfasst einige Komponente und ist sehr umständlich.

Zu installieren:

- Python
- Pip
- Salt
- Napalm
- Napalm-logs
- OATS
- OATSDBhelpers
- Datenbank

Hier wären verschiedene Lösungen denkbar von einfachen Installationsskripts, bis zur kompletten Verpackung in eine Installationssoftware. Unter den gängigen Linux-Distributionen können diese Komponente direkt aus der Kommandozeile heruntergeladen und installiert werden, dies könnten wir direkt in ein Skript verpacken.

Zu konfigurieren:

- Napalm-logs
- Salt Proxies
- Netzwerkgeräte

Eine Automatisierung der Konfiguration ist kaum vorstellbar da man hier vor allem die Daten des eigenen Netzes in die jeweiligen Konfigurationsfiles eintragen muss und diese für jedes Netzwerk unterschiedlich sein wird.



1.5.6 User Interface

Zurzeit kann nur mittels CLI mit OATS interagiert werden. Ein GUI würde OATS benutzerfreundlicher machen.

Stand SA

Während der Studienarbeit wurde ein CLI-basiertes UI für OATS entwickelt sodass die Use Cases erfüllt waren. So existieren zurzeit CLI-Befehle für folgende Abfragen:

- CRUD für Problemfälle
- CRUD für Netzwerkgeräte
- CRUD für Techniker
- Anzeigen des Netzwerkzustands (als zurzeit offene Problemfälle)

Generell lässt sich das User-Interface in zwei Teile aufteilen, die CRUD-Operationen und das Anzeigen des aktuellen Netzwerk-Zustandes (bzw. Network-Health).

Network-Health

Für das Anzeigen des aktuellen Netzwerkzustands würde sich eine Art Dashboard anbieten, die aktuelle Informationen zum Netzwerk grafisch darstellt. Folgende Informationen wären dabei von besonderem Interesse:

- Anzahl der offenen Problemfälle
- Anzahl der gelösten Problemfälle in einem bestimmten Zeitraum (z.B. während des letzten Tags)
- Aktuell empfangene Events (z.B. durch Syslog oder durch Streaming-Telemetry Evaluation)
- Aktuelle Streaming-Telemetry Daten
- Liste der Events auf die gehorcht wird
- Liste der am meisten aufgetretenen Events
- Zusammenfassung des aktuellen Zustands (z.B. als Ampel)

CRUD

Hier bietet sich eine einfache Webpage mit Inputfeldern für die verschiedenen Kategorien in der Datenbank an. Auf einer weiteren Webpage könnten dann die aktuellen Daten der Datenbank angezeigt werden.

Je von Domain zu Domain gilt es zu überprüfen, ob das Zugreifen auf die Datenbank nur nach einem Login möglich sein soll. Im Idealfall können sowieso nur Techniker/System-Administratoren auf das lokale Netz zugreifen, dies ist aber nur selten der Fall.

Welcome Page

Es ist sinnvoll, eine Welcome Page anzubieten, die eine Übersicht über OATS liefert und Links auf die CRUD-Operationen bzw. das Dashboard anbietet. Hier könnte evtl. auch ein Rendering der Netzwerktopologie der OATS zugrunde liegt, angezeigt werden.

Workflow CRUD

Für die Troubleshooting-Workflows wäre die letzte Stufe, eine Webpage anzubieten, in der ein Workflow aus mehreren vordefinierten Operationen «zusammengeklickt» werden kann. Es wurde aber bereits in der Studienarbeit argumentiert, dass nur sehr limitierte Workflows auf diese Weise umgesetzt werden könnten, die wahrscheinlich kein sinnvolles Troubleshooting ermöglichen.

Der Grund dafür ist, dass für praktisch jeden neuen Troubleshooting-Fall wieder komplett neue Operationen ins System eingefügt werden müssen, was das Anbieten einer grafischen Nutzeroberfläche sinnlos macht, da diese neuen Operationen sowieso zuerst programmiert werden müssen.



1.5.7 Weitere Troubleshooting-Fälle

Da das Projektteam keine Arbeitserfahrung in Campus- oder Data-Center-Netzwerken hat, wurden in einer Sitzung mit Prof. Laurent Metzger und Urs Baumann vom INS mögliche Probleme in Netzwerken diskutiert, die durch automatisches Troubleshooting behandelt werden könnten.

BGP

Grössere Firmen verfügen über eigene BGP-Router die eine grosse Anzahl von verschiedenen Routen (= IP-Prefixes) speichern können müssen (>100'000). Diese werden nur für einen bestimmten Zeitraum gespeichert und werden dann gelöscht, bis sie wieder vom Internet Provider angekündigt werden.

Das Problem

Es kommt vor, dass der Internet Provider von Zeit zu Zeit interne Probleme hat und dann seinen Verkehr nur ineffizient routen kann. Die Firma die den Internetzugang von diesem Provider bezieht bekommt dies nur durch eine langsame Verbindung mit, weiss aber nicht, dass der Ursprung des Problems **nicht** im eigenen Netz liegt.

Detektierung

Der Ursprung des Problems kann ermittelt werden, indem die Anzahl der gespeicherten Routen auf dem BGP Router kontrolliert wird. Sinkt die Anzahl in kurzer Zeit um einen bestimmten Wert (z.B. >10%), dann liegt ein Problem beim Provider vor und es können intern Massnahmen getroffen werden.

Massnahmen

Im Idealfall existiert ein zweiter Internetzugang über einen anderen Provider. Dann kann ein Failover stattfinden und der gesamte Verkehr über diesen Provider geleitet werden, bis der andere Provider seine Probleme behoben hat.

Existiert kein zweiter Internetzugang, so kann intern ein «Alarm» ausgelöst werden, der anzeigt, dass das Problem externer Natur ist und nicht durch eigene Massnahmen behoben werden kann. Anhand dieses Alarms kann man dann intern entscheiden wie man mit dem langsameren Internetzugang umgehen will, eine Möglichkeit wäre hier, die Geschwindigkeit bereits intern für alle Benutzer zu drosseln um zu verhindern, dass zwischen eigenem Netz und ISP ein Stau entsteht.

Umsetzung in OATS

1. Die Anzahl gespeicherter Routen wird regelmässig per Streaming-Telemetry an OATS geschickt.
2. In einem neuen Client innerhalb von OATs werden die aktuellen Daten evaluiert.
3. Beim Überschreiten eines vorbestimmten Schwellenwerts (z.B. Verlust von 10% der Routen in <5min), wird ein Event erzeugt
4. Als Reaktion auf das Event wird ein Failover auf einen anderen Provider durchgeführt (falls vorhanden).

Wireless

In einer Firma die über WLAN verfügt, existieren meistens mehrere WLAN-Access-Points über welche die Benutzer mit ihrem Mobilgerät Internetzugang beziehen können. Diese Access-Points sind wiederum mit einem Wireless-Controller verbunden welcher für die Konfiguration der Access-Points verantwortlich ist.



Das Problem

Es kommt vor, dass ein Access-Point seine Verbindung zu dem Wireless-Controller verliert und von selbst nicht mehr aufnehmen kann. Der Access-Point ist damit nutzlos.

Detektierung

Ein automatisches Detektieren ist hier kaum möglich, da nur die Endbenutzer merken, dass sie über keine Internetverbindung mehr verfügen.

Massnahmen

Der Access-Point kann remote runter- und wieder hochgefahren werden. Dies sollte das Problem beheben. Besteht das Problem weiterhin, so muss ein Techniker beauftragt werden.

Umsetzung in OATS

Da noch unklar ist, wie das Problem detektiert werden kann, ist auch die Umsetzung in OATS noch nicht klar.

Campus: QoS

Campus-Netze verfügen meistens über ein VoIP System. Der VoIP-Verkehr ist oft durch QoS priorisiert.

Das Problem

Es kommt vor, dass der Buffer eines Routers/Switches voll ist und somit anfängt Pakete zu dropen. Dies wirkt sich erheblich auf VoIP aus, da dann kein Gespräch mehr möglich ist.

Detektierung

Der Buffer des Routers kann kontinuierlich ausgelesen werden und sobald der Buffer voll ist kann eine Meldung ausgelöst werden, um frühzeitig zu erkennen, dass Pakete gedroppt werden könnten.

Massnahmen

Die Ursachen für das Problem können vielseitig sein und müssten zuerst durch einen Troubleshooting-Workflow ermittelt werden, bevor Massnahmen getroffen werden können.

Umsetzung in OATS

1. Regelmässiges Senden der Buffergrössen an OATS durch Streaming-Telemetry
2. Evaluieren der Daten in einem Client innerhalb von OATS
3. Beim Überschreiten eines bestimmten Schwellenwerts wird ein Event ausgelöst.
4. Als Reaktion auf das Event, könnte z.B. der Traffic für eine gewisse Zeit um den Router umgelenkt werden.

Datacenter

In einem Datacenter werden gewisse Operationen (z.B. Backups) nachts durchgeführt, damit das Netz zu Peak-Zeiten nicht unnötig belastet wird.

Das Problem

Manchmal wird durch interne Quellen auch tagsüber hoher Verkehr verursacht, sei es mutwillig oder aus Versehen. Dies führt zu einer allgemeinen Verlangsamung des Netzes, in einer Zeit in der es dies nicht verkraften kann.



Detektierung

Es existieren Tools, mit denen der Verkehr in Echtzeit analysiert werden kann (z.B. Netflow). Damit kann der Ursprung des hohen Verkehrsaufkommens (am besten gleich der verantwortliche Port) ermittelt werden.

Massnahmen

Kennt man den Port, der für den Verkehr verantwortlich ist, so kann der maximale Durchsatz auf diesem Port gedrosselt werden. So wird zwar die Operation auf diesem Port verlangsamt, aber die allgemeine Netzwerklast wird massiv verringert.

Umsetzung in OATS

1. Analysieren des Verkehrs im Netz in Echtzeit durch Netflow, Netflow sendet diese Daten an OATS.
2. Die Daten werden in einem Client innerhalb von OATS evaluiert und sobald ein Schwellenwert überschritten wird, wird ein Event erzeugt.
3. OATS sammelt Daten im Netz, um den Ursprung (=Port) des zu hohen Verkehrsaufkommens zu detektieren.
4. OATS drosselt den Verkehr am betroffenen Port.

Dauerhafte Netzwerkverkehrsspiegelung

Ein IDS analysiert den Netzwerkverkehr einer Firma in Echtzeit. Zu diesem Zweck wird ein Grossteil des gesamten Netzwerkverkehrs gespiegelt (kopiert und an das IDS geschickt). Das IDS führt dann verschiedene Operationen auf diesen Paketen, um z.B. zu erkennen dass eine DDoS Attacke stattfindet.

Das Problem

Die Spiegelung des Verkehrs führt zu einer hohen Netzwerklast, die nur selten einen Nutzen bringt, nämlich genau dann, wenn eine Attacke stattfindet.

Detektierung

Um eine DDoS Attacke zu erkennen, könnte z.B. per Streaming-Telemetry auf einem Router die Anzahl der eingehenden Anfragen von unterschiedlichen IP-Adressen überwacht werden. Überschreitet dieser Wert einen Schwellenwert, so wird ein Event ausgelöst, welcher dazu führt, dass ab diesem Zeitpunkt der Netzwerkverkehr an das IDS gespiegelt wird. So wird der Netzwerklast nur dann verdoppelt, wenn es wirklich nötig ist.

Massnahmen

Ist das Event an das IDS ausgelöst, wird die Behandlung der Attacke selbst dem IDS überlassen.

Umsetzung in OATS

1. Regelmässiges Senden der Anzahl Anfragen in z.B. den letzten 10 Sekunden durch Streaming-Telemetry.
2. Evaluieren der Daten auf einem Client innerhalb von OATS.
3. Überschreitet die Anzahl Anfragen einen Schwellenwert, so wird ein Event ausgelöst.
4. Als Reaktion auf das Event löst OATS eine Spiegelung des Netzwerkverkehrs an das IDS aus.



OSPF

OSPF ist ein Link-State-Routing Protokoll, welches oft in Campus- und Datacenter-Netzen eingesetzt wird, um IP-Verkehr zu routen.

Das Problem

Es kommt vor, dass ein OSPF-Prozess andauernd hoch- und runterfährt und somit für Chaos in dem vorhandenen OSPF-Netz führt, da andauernd neue Routen berechnet werden.

Detektierung

Das Flapping kann über Syslog detektiert werden, sollte also relativ einfach mit dem vorhandenen OATS System gelöst werden können.

Massnahmen

Das Problem kann durch Runter- und Hochfahren des betroffenen OSPF-Prozesses behoben werden.

Umsetzung in OATS

1. Bei jedem Hoch-/Runterfahren eines OSPF-Prozesses wird eine Syslog-Meldung an OATS ausgelöst.
2. OATS erkennt, wenn viele solcher Meldungen in kurzer Zeit ankommen, dass es sich dann um Flapping handelt.
3. OATS fährt den betroffenen Prozess runter und wieder hoch.

Spanning-Tree

Das Spanning-Tree-Protokoll ist ein Layer 2 Protokoll welches Broadcast-Stürme in einer Broadcast-Domain verhindert, indem es gewisse Pfade blockiert.

Das Problem

Manchmal kommt es vor, dass ein Port immer wieder hoch und runterfährt (Port-Flapping → siehe Problemanalyse). Dies führt wiederum zu Chaos beim Spanning-Tree-Protokoll, was das gesamte Netz verlangsamt.

Detektierung

Auch das Problem sollte relativ einfach durch Syslog detektiert und somit durch OATS behandelt werden können.

Massnahmen

Das betroffene Gerät kann runtergefahren und nach einer gewissen Zeit (damit sich STP wieder einpendeln kann) wieder hochgefahren werden. Dies sollte das Problem gelöst haben. Sollte das Problem weiterhin bestehen ist wohl weiteres menschliches Troubleshooting nötig.

Umsetzung in OATS

1. Bei jedem Runter-/Hochfahren eines Ports wird eine Syslog-Meldung an OATS ausgelöst.
2. OATS erkennt, wenn viele solcher Meldungen in kurzer Zeit ankommen, dass es sich dann um Port-Flapping handelt.
3. OATS fährt den betroffenen Router runter und nach z.B. 5 Minuten wieder hoch.



1.5.8 Weiterführende Themen

Während der Zwischenpräsentation mit dem Experten und den Betreuern wurden noch zwei Punkte angesprochen, die wir bis dahin noch nicht berücksichtigt bzw. behandelt hatten. Diese Punkte sollen hier der Vollständigkeit halber noch erwähnt werden.

Interaktion mit dem System

Es wäre denkbar mit OATS auch ein interaktives Troubleshooting durchzuführen. D.h. dass das Troubleshooting nicht einfach durchgeführt wird, sondern dass der Techniker gefragt wird, ob eine Lösung wirklich ausgeführt werden soll.

So könnte, wenn das System erkennt, dass ein Interface im «down» Status ist, der Techniker über eine Chatops-Applikation gefragt werden, ob das Interface wieder hochgefahren werden soll. Der Techniker kann dann per Knopfdruck bestätigen, was gemacht werden soll.

Fehlerklassifizierung

Die Fehlerklassifizierung wurde während des Projekts nur oberflächlich angeschnitten. So wird angenommen, dass wenn von unserem System ein Fehler im Netz entdeckt wird, dass dieser schlimm genug ist, um ein automatisches Troubleshooting zu rechtfertigen.

Es wäre aber auch denkbar unterschiedliche Klassen von Fehlern einzuführen (ähnlich wie die Severity-Levels bei Syslog), um zwischen Fehlern zu unterscheiden, welche ein Troubleshooting benötigen und solchen bei welchen z.B. ein einfacher Log-Eintrag genügt



2. Anforderungen

Änderungsgeschichte

Datum	Version	Änderung	Autor
28.09.2017	1.0	Erstellung des Dokuments	nv
30.09.2017	1.1	Überarbeiten Anforderungen Erstellen NFR	nv
02.10.2017	1.2	Priorisierung der Anforderungen	nv, rj
16.10.2017	1.3	Updaten Use Cases	rj
16.10.2017	1.4	Überarbeitung nach der Construction	rj
02.03.2018	2.0	Übernahme Dokument in BA	nv
08.03.2018	2.1	Update für Stand BA	nv, rj
15.03.2018	2.2	Update Systemumfang	nv, rj
20.03.2018	2.3	Überarbeitung nach Sitzung	nv, rj
26.03.2018	2.4	Überarbeitung nach Sitzung 2	nv, rj
01.06.2018	2.5	Überarbeitung vor Abgabe	rj
05.06.2018	2.6	Überarbeitung vor Abgabe	nv



2.1 Zweck

2.1.1 Beschreibung

Dieses Dokument beschreibt die Anforderungen an das automatische Troubleshooting System, welches im Rahmen der Studienarbeit von Nico Vinzens und Raphael Jöhl im HS2017 erarbeitet wurde und im FS2018 im Rahmen der Bachelorarbeit erweitert wird.

2.1.2 Gültigkeitsbereich

Das Dokument ist gültig während der Entwicklung des automatisierten Troubleshooting Systems OATS.

2.1.3 Übersicht

In Kapitel 2.2 wird der Umfang des Systems erläutert. Darin werden unter anderem die Systemfunktionalität, das Systemumfeld und die Architektur beschrieben.

Im Kapitel 2.3 folgen die Use Case Diagramme und die einzelnen Use Cases. Darauf folgen im Kapitel 2.4 die Anforderungen an die Workflows.

Das letzte Kapitel dokumentiert die nichtfunktionalen Anforderungen an das System.



2.2 Systemumfang

2.2.1 Stakeholder

- Betreuer der Arbeit: Sie bewerten den Ausgang der Arbeit und evaluieren das Endprodukt.
- Das Projektteam: Entwirft das System und ist für das Endprodukt verantwortlich.
- Systems-Engineers: potenzielle Benutzer des Systems, z.B. Systems Engineer eines Campus-Netzes eines KMUs

2.2.2 Systemfunktionalität

Das System dient zum automatischen Troubleshooting von Problemen, die innerhalb eines Netzwerkes vorkommen.

2.2.3 Systemumfeld

Das System wird im Rahmen eines Campus-Netzes eingesetzt. Wir arbeiten mit zwei Netzwerken. Das erste Netzwerk verfügt über je einen Access-, Distribution- und Core-Layer. Auf dem Core- und Distribution-Layer wird OSPF als Routing Protokoll eingesetzt. Beim zweiten Netzwerk haben wir zwei Switches die jeweils einen angeschlossenen Computer haben.

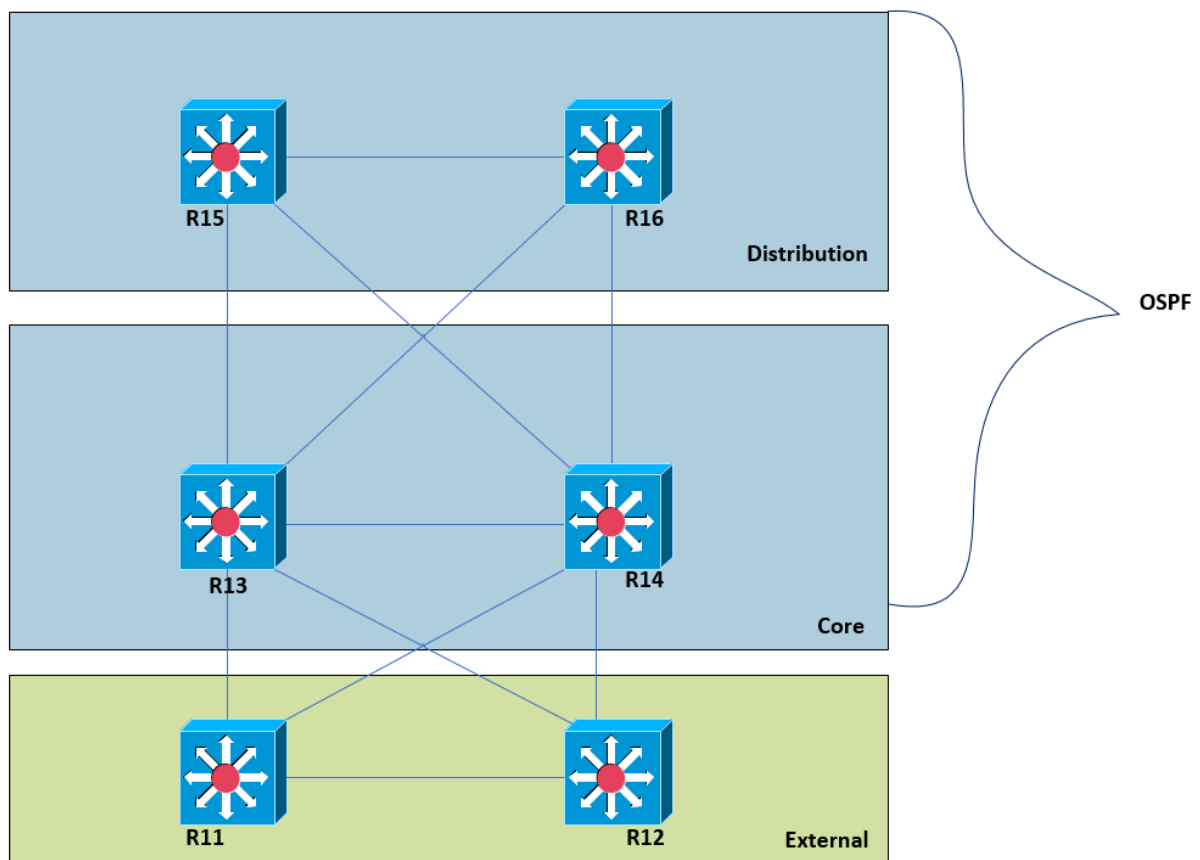


Abbildung 25: Campus-Netz

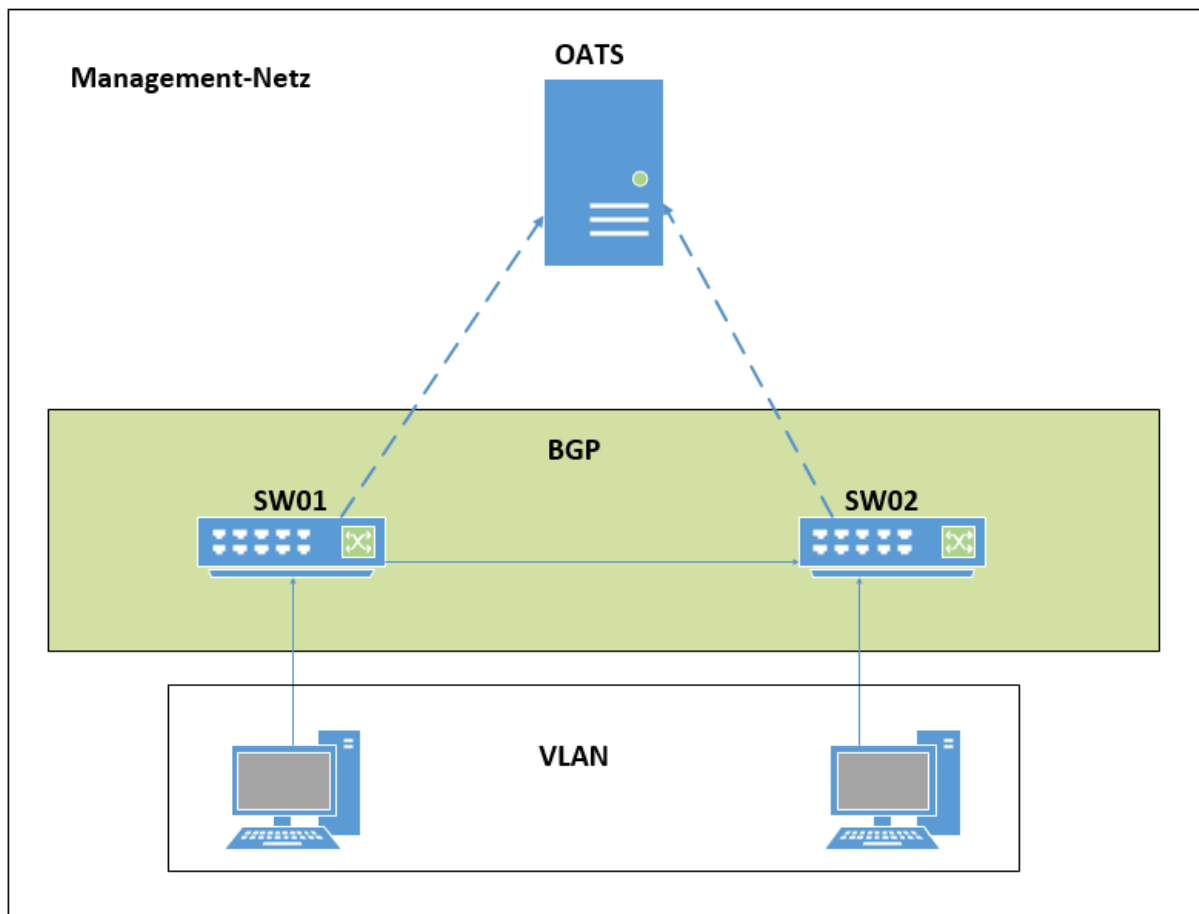


Abbildung 26: Campus-Netz mit BGP und VLAN

2.2.4 Architekturbeschreibungen

Da der Bachelorarbeit ein bereits bestehendes System (OATS) zugrunde liegt, wird hier aufgezeigt, wie die Architektur von OATS während der Elaboration-Phase der Bachelorarbeit aufgebaut war.

An dieser Stelle wird nur kurz die Rolle der einzelnen Tools aufgezeigt, für genauere Beschreibungen zu OATS und für eine Analyse der einzelnen Tools wird auf das Architekturkapitel (Kapitel 3) dieses Dokuments verwiesen.

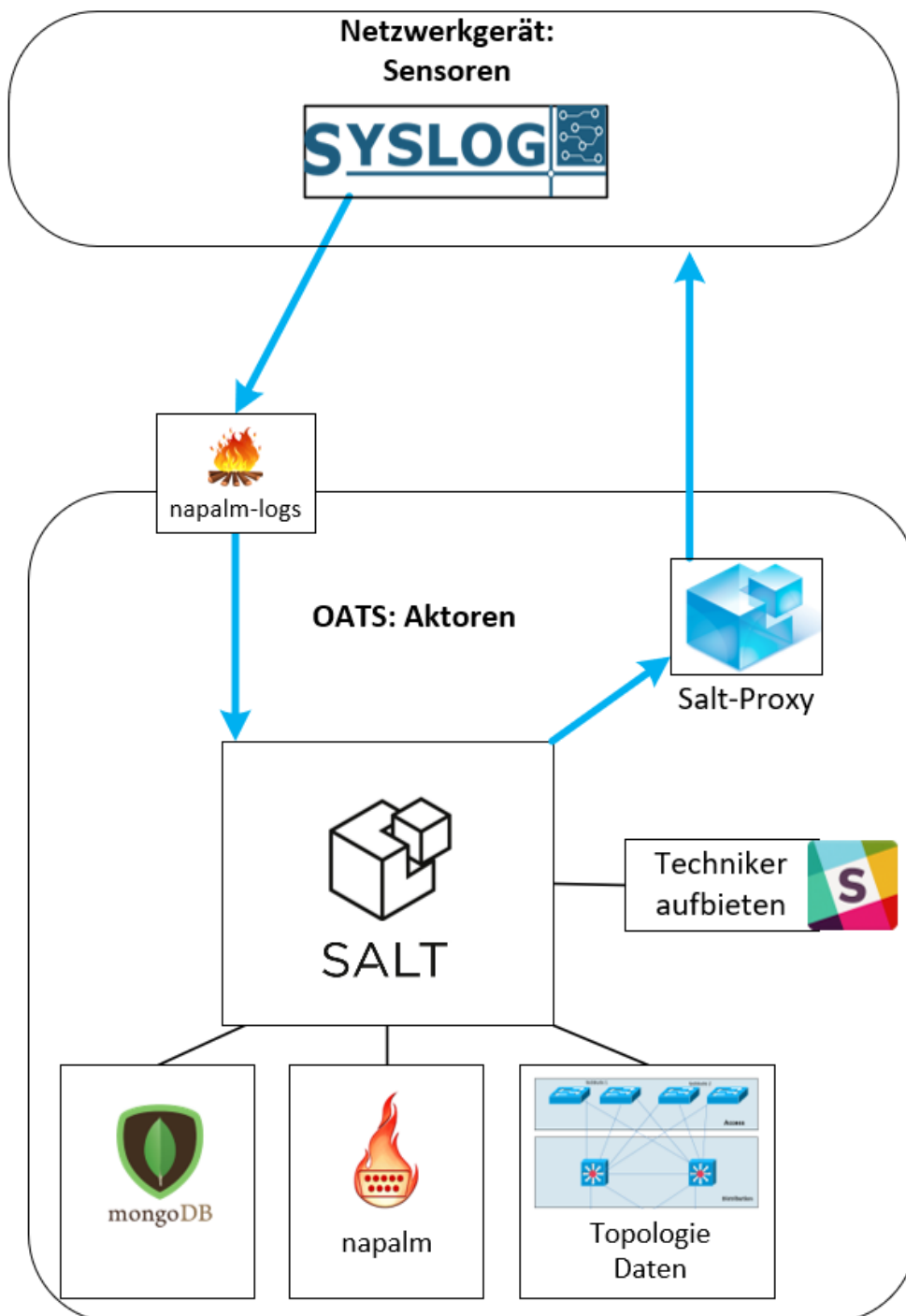


Abbildung 27: Architekturübersicht

Syslog

Syslog dient als Sensor für mögliche Troubleshooting-Fälle [1]. Syslog wird auf jedem Netzwerkgerät manuell eingerichtet. Es muss genau definiert werden, welche Meldungen geschickt werden und welche nicht. Syslog schickt dann die definierten Meldungen an OATS. Da die Syslog Meldungen keine definierte Form haben, müssen sie vom System selber in Modellform transformiert werden.

Napalm-logs

Napalm-logs ist eine Open Source Software, die als Daemon auf dem OATS-Server läuft und auf Syslog Meldungen horcht und diese in eine herstellerunabhängige Form bringt (und somit im Code



verarbeitbar macht). Sind die Daten verarbeitet, so werden sie zusammen mit einem Event an den Salt-Event-Bus übermittelt

Salt

Salt ist ein verteiltes Konfigurations-Management-Tool, welches in OATS integriert ist und das Herzstück der Applikation bildet. Es ist für folgende Aufgaben zuständig:

- Ausführen von Workflows als Reaktion auf ein Event, welches von napalm-logs ausgelöst wurde.
- Sammeln von Informationen:
 - Über DB-Queries
 - Über direkte Interaktion mit den Netzwerkgeräten (mittels napalm)
- Ausführen von Massnahmen zum Beheben von Problemen (mittels napalm)
- Benachrichtigen von Technikern bei nicht behebbaren Problemen (mittels Slack)

Salt-Proxy

Der Salt-Proxy ist zuständig für die Kommunikation mit den Netzwerkgeräten. Für jedes Gerät im Netz existiert ein solcher Proxy, welcher jeweils wieder mit dem Salt-Master kommuniziert.

Napalm

Napalm (Network Automation and Programmability Abstraction Layer with Multivendor support) ist eine Open Source Python-Library, die Funktionen implementiert, welche es einem erlauben mit unterschiedlichen Netzwerkgeräten zu interagieren (herstellerunabhängig). So ermöglicht napalm den Verbindungsaufbau zu Geräten, die Manipulation deren Konfiguration und das Sammeln von Informationen. Im Falle von OATS ist napalm direkt in Salt integriert.

MongoDB

MongoDB ist eine NoSQL-Datenbank in welcher folgende Daten abgelegt sind:

- Topologie-Daten
- Daten zu Problemfällen
- Daten zu Technikern

Slack

Falls ein Problem auftaucht welches nur von einem Techniker vor Ort gelöst werden kann, muss man diesen aufbieten. Diese Aufgabe wird vom Instant-Messaging-Dienst Slack übernommen, welcher auch in Salt integriert ist.

2.2.5 Einschränkungen

Das automatische Troubleshooting beschränkt sich auf ein lokales Netz. Es wird nicht dafür ausgelegt verteilte Netze als Ganzes überwachen zu können.

Das System wird in einem Netz entwickelt in dem nur Cisco-Geräte eingesetzt werden. Theoretisch sollte das System durch die eingesetzten Tools herstellerunabhängig sein, ist aber nicht garantiert, da einige der Funktionen z.B. nur für Cisco implementiert wurden, aber noch nicht für Juniper.

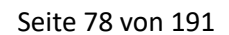
2.2.6 Annahmen

Die Netzwerktopologie muss bereits erfasst und an einer zentralen Stelle abgelegt sein. Die Daten werden dann von OATS über eine Schnittstelle eingelesen.



2.2.7 Abhängigkeiten

Die eingesetzten Geräte müssen mit napalm kompatibel sein. Eine Liste der unterstützten Geräte von napalm findet man unter [12].



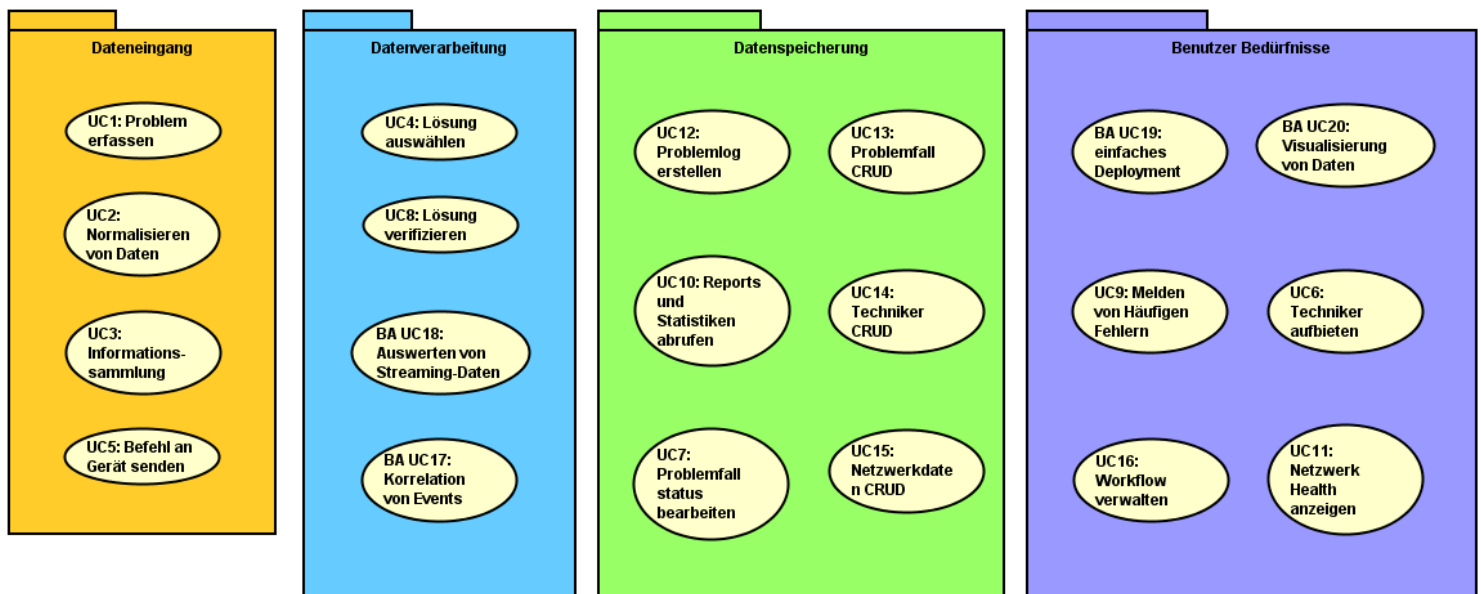


Abbildung 29: Use Case Diagramm nach Kategorien

2.3.3 Use Case Diagramm nach Kategorien

Die Use Cases können allgemein in vier Kategorien aufgeteilt werden:

- Dateneingang: Use Cases welche mit der Kommunikation zwischen System und Netzwerkgeräten, um etwa Daten zu sammeln, zu tun haben.
- Datenverarbeitung: Use Cases welche mit der Auswertung der erhaltenen Daten aus dem Netzwerk zu tun haben.
- Datenspeicherung: Use Cases welche mit der Speicherung von Daten zu tun haben, entweder als Ressourcen, als Zwischenspeicher oder als Archivierung.
- Benutzer Bedürfnisse: Use Cases die damit zu tun haben die Bedürfnisse der Nutzer des Systems zu decken, etwa eine Visualisierung der Daten.

2.3.4 Use Cases im Brief Format

Anmerkung: Use Cases welche bereits in der Studienarbeit erfasst wurden sind mit SA gekennzeichnet, während neue Use Cases mit BA (Bachelorarbeit) markiert sind.

SA UC1: Problemfall erfassen

Es sind 3 mögliche Arten möglich, wie ein Problemfall erfasst werden kann:

SA: Aktiver Sensor

Ein aktiver Sensor sendet als Reaktion auf ein Event eine Meldung an OATS und startet so den Troubleshooting-Prozess. Bereits in der Studienarbeit implementiert wurde Syslog. Eine weitere Möglichkeit wäre das Einbinden von NETCONF.

BA: Passiver Sensor

Ein passiver Sensor sammelt kontinuierlich Daten im Netz und sendet diese an OATS. Diese Daten werden innerhalb von OATS evaluiert/korreliert und reagiert aufgrund der Ergebnisse eventuell mit einem Troubleshooting-Prozess.



SA: Manuelle Problemerkfassung

Alternativ kann ein Systemadministrator oder ein Techniker manuell ein Problemfall erstellen, etwa um eine Überprüfung des Systems durchzuführen.

SA UC2: Normalisieren von Daten

Damit mehrere Quellen zum Erhalten von Daten verwendet werden können müssen diese normalisiert werden.

SA/BA UC3: Informationen von Geräten sammeln

Das System muss in der Lage sein, Informationen von einem Gerät im Netzwerk abzufragen und diese dem richtigen Problemfall zuzuordnen oder einen neuen Problemfall erstellen. Falls ein Problem nicht genau identifiziert werden konnte muss das System von einem Gerät auf Anfrage noch weitere Daten erhalten. Dazu wird hier napalm und Salt verwendet.

Zudem kann auf historische Daten zugegriffen werden. So kann ermittelt werden, ob der Zustand des Netzes von der Norm abweicht.

SA UC4: Lösung auswählen

Aufgrund der gesammelten Informationen entscheidet sich das System für eine Lösung um das ermittelte Problem zu beheben. Falls das System das Problem nicht selber lösen kann und ein Techniker aufgeboden wird, muss das System zudem entscheiden wie dringend sich dieser dem Problem annehmen muss.

SA UC5: Befehl an Geräte senden

Während dem Troubleshooting Prozess muss es dem System möglich sein, Befehle an ein Gerät, zu senden. Beispiele dazu sind System- und Konfigurationsdaten anfordern um das Problem zu identifizieren oder ein ausführbares Skript senden welches das Problem beheben sollte. Diese Kommunikation zu den Netzwerkgeräten wird über napalm geregelt.

SA UC6: Techniker aufbieten

Falls es dem System nicht möglich ist ein Problem automatisch zu identifizieren oder zu lösen muss ein Techniker aufgeboden werden, um sich diesem anzunehmen.

SA UC7: Problemfall Status bearbeiten

Ein Techniker oder ein Systemadministrator der an einem Problemfall arbeitet, muss in der Lage sein, Informationen zu diesem zu bearbeiten, etwa falls er das Problem abgeschlossen hat.

SA/BA UC8: Lösung verifizieren

Nachdem eine Lösung für ein Problem ausgeführt wurde, muss das System überprüfen ob das Problem nun gelöst wurde und somit den Fall abschliessen kann. Die Verifikation wird von Salt ausgeführt, wobei auf Informationen der betroffenen Geräte zugegriffen werden muss und somit wie bei Use Case 3 und 5 napalm und napalm-logs zur Kommunikation eingesetzt wird.

Zudem können die aktuellen Streaming-Daten mit dem SOLL verglichen werden um zu überprüfen ob der Netzwerkzustand wie gewünscht ist.

SA UC9: Melden von häufigen Fehlern

Falls oft dieselben Fehler auftreten, kann dies auf ein grundlegendes Problem im Netzwerk oder in dessen Aufbau hinweisen, das System muss in der Lage sein diese zu identifizieren und einen Systemadministrator zu benachrichtigen. Detektiert werden solche Häufigkeiten über die geloggtten Daten aus der Datenbank oder direkt in Salt. Der Systemadministrator wird über Slack auf dies aufmerksam gemacht.



SA UC10: Reports und Statistiken abrufen

Ein Systemadministrator möchte, aus Informationen von aufgetretenen Problemfällen, Statistiken und Reports zu einzelnen Geräten oder dem kompletten Netzwerk abrufen. Dazu soll er direkt eine Datenbank Abfrage machen können.

SA UC11: Network-Health anzeigen

Um schnell eine Übersicht über den Zustand des Netzwerks zu erhalten kann ein Techniker den Network Health Counter anzeigen lassen, welcher ihm Informationen über den Zustand des Netzwerks gibt. Dies kann man direkt über Salt lösen.

SA UC12: Problemlog erstellen

Sobald ein Troubleshooting-Prozess durch ein Event ausgelöst wird, wird in der DB ein Problemfall mit allen relevanten Details erstellt. Der Problemfall wird während des Troubleshooting-Prozesses aktualisiert und am Ende als abgeschlossen markiert.

SA UC13: Techniker CRUD

Ein Systemadministrator erstellt Techniker und verwaltet diese, damit sie bei einem Problemfall aufgeboden werden können.

SA UC14: Problemfall CRUD

Ein Systemadministrator kann einzelne Informationen zu Problemfällen lesen und bearbeiten. Er kann die Problemfälle auch selber erstellen oder löschen.

SA UC15: Netzwerkdaten CRUD

Ein Systemadministrator unterhält die relevanten Netzwerkdaten für das System in einem externen File.

SA UC16: Workflow verwalten

Falls neue Probleme auftauchen oder weitere Probleme automatisch gelöst werden sollen, muss ein Systemadministrator dazu in der Lage sein einen neuen Workflow im System zu erstellen. Existierende Workflows sollen einsehbar sein und wenn nötig soll es auch möglich sein diese zu editieren. Falls ein Workflow nichtmehr benötigt wird, etwa weil das betreffende Protokoll nichtmehr verwendet wird, soll man einen Workflow auch löschen können.



2.3.5 Zusätzliche Use Cases in der BA

BA UC17: Korrelation von Events

Das System ist in der Lage Events zueinander zu korrelieren und so den Ursprung eines Problems zu erkennen. In einem ersten Schritt sollen Events aus den gleichen Quellen korreliert werden. Später soll zudem möglich sein diese miteinander zu vergleichen.

Scope	System oder Programm zur Event-Korrelation
Pre-Condition	Sensoren im Netzwerk haben Events an unser System gesendet und unser System hat diese erhalten. SA UC1, UC2, UC3
Beschreibung	Innerhalb eines Workflows können die Events zusammengehängt und Probleme identifiziert werden.
Ziel	Erkennen eines Problems, welches mit einem einzelnen Event nicht erkannt wurde.
Beispielhafter Ablauf (regel-basierter Ansatz)	1. Das System erhält innerhalb von kurzer Zeit mehrere OSPF_NEIGHBOR_DOWN Events. 2. Das System erkennt, dass es sich dabei um zusammenhängende Events handelt. 3. Das System erkennt den Ursprung der Events (ein eingefrorener OSPF-Prozess) 4. Das System behebt den Fehler durch Runter- und Hochfahren des betroffenen Prozesses. 5. Das System erhält innerhalb von kurzer Zeit mehrere OSPF_NEIGHBOR_UP Events. 6. Das System erkennt, dass es sich bei diesen Events um die Reaktion des Netzes auf die durch das System getroffenen Massnahmen handelt und schliesst daraus, dass das Problem erfolgreich behandelt wurde.
Beispielhafter Ablauf (Machine Learning)	Pre-Condition 1: Es muss eine grosse Anzahl Daten über den Netzwerkverkehr verfügbar sein. Pre-Condition 2: Es muss ermittelt worden sein, welcher ML-Algorithmus am besten zu den gegebenen Daten passt. Ablauf: 1. Der Machine Learning Algorithmus lernt anhand der verfügbaren Daten und erkennt ein Muster mit welchem sich ein Fehler voraussagen lässt. 2. Das Muster wird in das System implementiert. 3. Es wird überprüft, ob das Muster die Fehler korrekt voraussagen kann. a. Das System kann die Fehler korrekt voraussagen. b. Das System kann die Fehler nicht korrekt voraussagen. ➔ Es müssen mehr Daten gesammelt werden, damit der ML-Algorithmus lernen kann, Fehler korrekt vorauszusagen.



BA UC18: Auswerten von Streaming-Daten

Das System ist in der Lage die erhaltenen Streaming-Daten (Daten welche in regelmässigen Abständen an das System gesendet werden) in Echtzeit auszuwerten und für eine persistente Speicherung und Auswertung in eine Datenbank zu schreiben.

Streaming-Daten können von verschiedenen Quellen übertragen werden, z.B. durch Streaming-Telemetry oder Netflow.

Dieser Use Case benötigt den überarbeiteten Use Case 3 „Informationen von Geräten sammeln“ aus der SA, welcher nun auch Streaming-Daten einschliesst.

Scope	System oder Programm zur Auswertung von Streaming Telemetry
Pre-Condition	Daten aus der Netzwerkumgebung wurden über einen Stream in das System eingelesen und abgespeichert. SA UC1, UC2, UC3
Beschreibung	Die erhaltenen Daten aus dem Netzwerk wurden überprüft und verglichen.
Ziel	Anstoss eines Workflows zur Problembehandlung oder Entwarnung, das kein Problem vorhanden ist.

BA UC19: Einfaches Deployment

Das OATS System kann als Ganzes, mit all seinen Komponenten, auf eine einfache Art und Weise aufgesetzt werden.

Aktor	Systemadministrator
Pre-Condition	Ein Server oder eine VM mit genügend Speicherplatz muss vorhanden sein.
Beschreibung	Der Systemadministrator führt das Installationsprogramm aus welches ihn durch die Installation von OATS und seinen Komponenten führt.
Ziel	OATS und die benötigten Komponenten wurden installiert.

BA UC20: Visualisierung von Daten

Das System bietet ein Dashboard zur Visualisierung diverser relevanten Daten.

Aktor	Nutzer von OATS
Pre-Condition	OATS ist lauffähig und der Nutzer hat Zugriff darauf.
Beschreibung	Über eine Nutzeroberfläche kann man Informationen abrufen und Einstellungen ändern.
Ziel	Der Nutzer ist über den Zustand des Netzes, welches von OATS überwacht wird, informiert.



2.4 Workflow Anforderungen

Nach der Zwischenpräsentation der Bachelorarbeit, welche Anfangs der Construction Phase statt find, wurde entschieden, dass der Fokus der Arbeit nicht auf der Umsetzung von neuen Workflows liegen soll, sondern das möglichst einfache Umsetzen von neuen Workflows (als nicht funktionale Anforderung «Modifizierbarkeit» erfasst). Da jedes Unternehmen eigene Probleme in ihren Netzwerken hat, ist diese unserer Ansicht nach deutlich sinnvoller. Die folgenden Anforderungen wurden deshalb nicht oder nur teilweise umgesetzt.

2.4.1 Abhängigkeiten

Die neuen Anforderungen für die Bachelorarbeit haben ausserhalb der Abhängigkeit an andere Anforderungen noch folgende Abhängigkeiten:

- OATS: Funktionalitäten welche in OATs integriert sind
- BA-Use Cases:
 - Streaming-Telemetry
 - Event-Korrelation
- Unbekannt: Für Anforderungen für welche noch nicht bekannt ist, wie sie umgesetzt werden können.

2.4.2 BGP

Nr.	Anforderung	Abhäng
1	Das System erkennt, dass die Verbindung zu einem Internet Provider langsamer ist als sie sein sollte.	Streaming-Telemetry
2	Das System kontrolliert die Anzahl an gespeicherten Routen auf einem BGP Router.	Streaming-Telemetry
3	Das System führt einen Failover auf einen zweiten ISP durch, falls einer vorhanden ist.	Nr. 1, 2

2.4.3 Wireless

Nr.	Anforderung	Abhäng
4	Das System erkennt, wenn ein Wireless-Controller keine Verbindung mehr mit einem Wireless Access-Point hat.	Unbekannt
5	Das System stellt die Verbindung eines Wireless-Controllers zu einem AP wieder her.	Nr. 4

2.4.4 Campus QoS

Nr.	Anforderung	Abhäng
6	Das System liest den Buffer eines Routers aus.	Streaming-Telemetry
7	Das System leitet den VoIP Verkehr um (umgeht den betroffenen Router).	OATS

2.4.5 Datacenter

Nr.	Anforderung	Abhäng
8	Das System erkennt, wenn der Verkehr im Netz besonders hoch ist.	OATS
9	Das System analysiert die Daten über die Höhe des Netzwerkverkehrs.	OATS
10	Das System drosselt den Verkehr an einem bestimmten Port	OATS



2.4.6 Dauerhafte Netzwerkverkehrsspiegelung

Nr.	Anforderung	Abhäng
11	Das System erkennt in Echtzeit, dass auf einem Port oder Gerät besonders viele Pakete von (verschiedenen) IP-Adressen erhalten werden.	Streaming-Telemetry
12	Das System leitet das Spiegeln des Verkehrs an ein IDS ein.	Nr. 11

2.4.7 OSPF

Nr.	Anforderung	Abhäng
13	Das System erkennt, dass ein OSPF Prozess dauernd hoch und runterfährt.	OATS
14	Das System fährt den OSPF-Prozess des betroffenen Routers runter und wieder hoch.	Nr. 13

2.4.8 Spanning-Tree

Nr.	Anforderung	Abhäng
15	Das System erkennt auf welchem Gerät Port-Flapping stattfindet.	OATS
16	Das System fährt das betroffene Gerät runter und nach einer bestimmten Zeitdauer wieder hoch.	Nr. 15

2.4.9 Event-Korrelation

Nr.	Anforderung	Abhäng
17	Das System korreliert Events anhand des Regel-basierten Ansatzes	-
18	Das System korreliert Events mithilfe von Machine Learning	-



2.5 Nicht funktionale Anforderungen

Die folgenden NFRs wurden der ISO 9126 Spezifikation entnommen [41].

2.5.1 Modifizierbarkeit

Das System kann einfach mit weiteren Funktionen ergänzt werden. So kann, wenn das System im produktiven Umfeld eingesetzt wird, einfach durch Module ergänzt werden, die die häufigsten Fehler im Netz automatisch beheben können.

Dies wird durch eine Development-Anleitung sichergestellt (siehe Kapitel 4.5).

2.5.2 Interoperabilität

- Das System ist unabhängig von der im Netzwerk eingesetzten Hardware.
- Das System ist unabhängig von den im Netzwerk eingesetzten Networking-Protokollen.

Es muss dabei nicht alle Protokolle und Geräte unterstützen, soll von deren Existenz aber nicht negativ beeinflusst werden.

Dies ist durch die Verwendung von Salt, napalm und napalm-logs sichergestellt (siehe Kapitel 3).

2.5.3 Richtigkeit

Eingriffe des Systems im Netzwerk belassen das System im schlimmsten Fall (→ Das System kann den Fehler nicht selbst beheben) im gleichen Zustand wie vor dem Eingriff.

Napalm garantiert, dass die Zugriffe atomar und idempotent sind, somit ist auch dies sichergestellt.

2.5.4 Skalierbarkeit

Das System funktioniert in kleinen Netzwerken gleich wie in grossen Netzwerken, dazu sollen die einzelnen Komponenten so gewählt werden, dass das System als Ganzes gut skaliert.



3. Architektur

Änderungsgeschichte

Datum	Version	Änderung	Autor
11.10.2017	1.0	Erstellung des Dokuments	rj, nv
14.10.2017	1.1	Stackstorm, Ansible, Logging	rj
15.10.2017	1.2	Salt, NAPALM, NAPALM-logs Pyangbind, ZeroMQ, Fazit etc.	nv
16.10.2017	1.3	Allgemeine Bearbeitung vor Abgabe	nv, rj
08.12.2017	2.0	Überarbeitung nach der Construction	rj
13.12.2017	2.1	Ergänzung nach Construction	nv
16.03.2018	3.0	Übernahme Dokument in BA	nv
04.06.2018	3.1	Überarbeitung nach Construction	nv
05.06.2018	3.2	Ergänzung nach Construction	nv, rj
06.06.2018	3.3	Datenspeicherung, Logische Architektur	nv, rj
07.06.2018	3.4	Architekturübersicht, Logische Architektur	nv, rj
08.06.2018	3.5	Deployment	nv, rj
09.06.2018	3.6	Korrektur	rj



3.1 Einführung

3.1.1 Zweck

Dieses Dokument beschreibt die Architektur von OATS zu zwei verschiedenen Zeitpunkten:

1. Zum Ende der Studienarbeit
2. Zum Ende der Bachelorarbeit

Das Dokument hat folgende Struktur:

Im ersten Teil werden Tools analysiert und verglichen, welche für die verschiedenen Use Cases von OATS in Frage kamen. Die Unterkapitel «Aktive Sensoren», «Aktoren», «Troubleshooting- und Konfigurationsmanagement-Tools», «ZeroMQ» stammen noch aus der Studienarbeit.

Danach wird die Architektur, zu den zwei Zeitpunkten beschrieben. Die nachfolgenden Kapitel befassen sich daraufhin nur noch mit der Bachelorarbeit.

So werden im Kapitel Abläufe mögliche Szenarien bei der Verwendung von OATS beschrieben. In der logischen Architektur wird auf die Implementation der einzelnen Tools eingegangen, während im Kapitel Deployment beschrieben wird, wie das Deployment der Komponenten von OATS auf einem Server aussieht. Im darauffolgenden Kapitel Datenspeicherung, wird beschrieben wie und in welchem Format, Informationen in OATS gespeichert werden. Das Kapitel Prozesse und Thread befasst sich damit welche Prozesse und Threads OATS beinhaltet. Schliesslich werden im Kapitel Grössen und Leistungen die Laufzeit von OATS analysiert.

3.1.2 Gültigkeitsbereich

Das Dokument ist gültig während der Entwicklung des automatisierten Troubleshooting Systems OATS.



3.2 Toolanalyse

In diesem Kapitel wird analysiert, wie die Use Cases, die in der Anforderungsspezifikation definiert worden sind, umgesetzt werden können. Da der Fokus der Arbeit darauf liegt, keine komplette Eigenlösung zu entwickeln, sondern mehrere bestehende Tools zu einem sinnvollen Ganzen «zusammenzuhängen», werden jeweils mehrere Tools beschrieben und miteinander verglichen.

3.2.1 Sensoren

Für die Use Cases, welche sich mit Dateneingang befassen, werden Sensoren benötigt. In OATS werden zwischen zwei Arten von Sensoren unterschieden, Zustandssensoren und Daten-Sensoren. Die Zustandssensoren lösen Events aus, sobald gewisse Ereignisse eintreten (z.B. ein Interface fährt runter).

Die Daten-Sensoren sammeln kontinuierlich Daten, schicken sie an OATS und lösen aber selbst keine Events aus. Die Daten müssen zuerst von einem nachgeschalteten Client analysiert werden, welcher dann die Events auslöst.

Die Sensoren werden hier nur kurz erklärt und verglichen, genauere Beschreibungen zu den Sensoren, findet man in der Domainanalyse.

Zustandssensoren

Syslog

Syslog wird auf jedem Netzwerkgerät manuell eingerichtet. Es muss genau definiert werden, welche Meldungen geschickt werden und welche nicht. Syslog schickt dann die definierten Meldungen an OATS. Da die Syslog Meldungen keine definierte Form haben, müssen sie vom System selber in Modellform transformiert werden.

Syslog wurde im Laufe der SA bereits als Zustandssensor in OATS integriert.

NETCONF

Der Subscription <> Notification Mechanismus von NETCONF erlaubt es, dass auf gewisse Events mit einer asynchronen Nachricht reagiert wird. Diese Nachrichten werden dann an eine zentrale Sammelstelle übermittelt. Anders als bei Syslog sind die Daten schon in XML, JSON oder YANG modelliert.

NETCONF würde in OATS eine ähnliche Rolle wie Syslog übernehmen und wird deshalb aussen vorgelassen, bis ein expliziter Use Case für das Einbinden von NETCONF spricht.

Es ist allerdings wahrscheinlich, dass NETCONF als Teil der Cisco-Streaming-Telemetry zum Einsatz kommt, da die Cisco IOS-XE Geräte in der Lab-Umgebung eine NETCONF-Session für Streaming-Telemetry voraussetzen.

Daten-Sensoren

Streaming-Telemetry

Streaming-Telemetry ist ein neuer Ansatz zur Netzwerküberwachung, in welchem kontinuierlich Daten von Geräten, in inkrementellen Updates, gestreamt werden (Push anstatt Pull). Eine zentrale Sammelstelle (OATS) kann dabei angeben, für welche Daten sie sich interessiert und erhält dann nur die gewünschten Daten. Zu diesen übertragbaren Daten gehören auch die Statistiken, die per IP SLA und Netflow gesammelt werden können (siehe unten).

Dieser Ansatz und die dazugehörige Datenverarbeitung und -speicherung, wurde bereits als fixer Use Case erfasst und wird deswegen in OATS integriert.



Netflow

Mittels Netflow können in Echtzeit Datenströme eines Netzes erfasst und exportiert werden. Damit kann erkannt werden, wenn der Verkehr im Netz zu gross wird. Dies erfüllt einige der Workflow Anforderungen (siehe Anforderungsspezifikation) und soll deswegen in OATS integriert werden. Die Streaming-Telemetry Recherche hat dabei gezeigt, dass die Flow-Statistiken als Bestandteil der Streaming-Telemetry übertragen werden können, somit ist das Problem der unterschiedlichen Datenformate zwischen Netflow und Streaming-Telemetry gelöst.

IP SLA

Internet Protocol Service Level Agreement oder kurz IP SLA ist ein Protokoll von Cisco IOS, mit welchem es möglich ist, Informationen über Network Performance in Echtzeit zu sammeln. Dieses Monitoring wird über ein aktives Verfahren erstellt, d.h. ein IP SLA Router hat die Fähigkeit, Traffic zu generieren und diesen direkt zu überprüfen.

Die SLAs sammeln Daten über Antwortzeit, Latenz, Jitter, Paketverluste, Connectivity, Server oder Webseiten Antwortzeit und Downtime, sowie Pfad- und Paket-Sequenzierung. Mit diesen Werten ist es möglich, grundsätzliche Informationen über die Netzwerkperformance zu sammeln. Zusätzlich ist es so möglich, die Quality of Service überprüfen und man sieht wo die Performance zu niedrig ist. Auch die IP SLA Daten können, per Streaming-Telemetry, kontinuierlich an OATS gesendet werden.

Fazit

Da mit Syslog bereits ein aktiver Sensor in OATS integriert wurde, ist es logisch, dass mindestens ein passiver Sensor (Streaming-Telemetry) hinzugefügt wird.

Eine wichtige Frage wird der Umgang mit den Daten sein, denn es soll in Zukunft möglich sein, Syslog-Daten mit Streaming-Telemetry-Daten zu korrelieren. Zu diesem Zweck muss ein einheitliches Datenformat definiert werden.

Im Verlaufe der Arbeit hat sich herausgestellt, dass auf den Netzwerkgeräten im Test-Lab, die Streaming-Telemetry Modelle z.T. nicht komplett implementiert wurden. So war es z.B. nicht möglich, die IP-SLA Daten, oder Netflow über Streaming-Telemetry auszulesen.



3.2.2 NAPALM

Napalm (Network Automation and Programmability Abstraction Layer with Multivendor support) ist eine Open Source Python-Library, die Funktionen implementiert, welche es einem erlauben, mit unterschiedlichen Netzwerkgeräten zu interagieren. So ermöglicht napalm den Verbindungsaufbau zu Geräten, die Manipulation deren Konfiguration und das Sammeln von Informationen.

Im Rahmen des Projekts wird napalm vor allem als API zum Sammeln von Informationen benutzt, um Entscheidungen über den weiteren Verlauf des Troubleshootings treffen zu können [42].

Unterstützte Betriebssysteme

Napalm unterstützt folgende Betriebssysteme:

- Arista EOS
- Cisco IOS
- Cisco IOS-XR
- Cisco NX-OS
- Fortinet Fortios
- IBM
- Juniper JunOS
- Mikrotik RouterOS
- Palo Alto NOS
- Pluribus
- Vynos

Somit kann eine Vielzahl von Geräten in einem Netzwerkkumfeld angesprochen werden, was die Flexibilität des Systems um ein Vielfaches erhöht. Napalm wurde bereits während der Studienarbeit in das System integriert.



Unterstützte Methoden

	EOS	FORTIOS	IOS	IOSXR	JUNOS	NXOS
connection_tests	{}	{}	{}	{}	{}	{}
get_arp_table	✓	✗	✓	✓	✓	✓
get_bgp_config	✓	✗	✗	✓	✓	✗
get_bgp_neighbors	✓	✓	✓	✓	✓	✓
get_bgp_neighbors_detail	✓	✗	✗	✓	✓	✗
get_config	✓	✓	✓	✓	✓	✓
get_environment	✓	✓	✓	✓	✓	✗
get_facts	✓	✓	✓	✓	✓	✓
get_firewall_policies	✗	✓	✗	✗	✗	✗
get_interfaces	✓	✓	✓	✓	✓	✓
get_interfaces_counters	✓	✓	✓	✓	✓	✗
get_interfaces_ip	✓	✗	✓	✓	✓	✓
get_lddp_neighbors	✓	✗	✓	✓	✓	✓
get_lddp_neighbors_detail	✓	✗	✓	✓	✗	✓
get_mac_address_table	✓	✗	✓	✓	✗	✓
get_network_instances	✓	✗	✗	✗	✓	✗
get_ntp_peers	✗	✗	✗	✓	✓	✓
get_ntp_servers	✓	✗	✓	✓	✓	✓
get_ntp_stats	✓	✗	✓	✓	✓	✓
get_optics	✓	✗	✓	✗	✓	✗
get_probes_config	✗	✗	✗	✓	✓	✗
get_probes_results	✗	✗	✗	✓	✓	✗
get_route_to	✓	✗	✗	✓	✓	✗
get_snmp_information	✓	✗	✓	✓	✓	✓
get_users	✓	✗	✗	✓	✓	✓
is_alive	✓	✓	✓	✓	✓	✓
ping	✓	✗	✓	✗	✓	✗
post_connection_tests	{}	{}	{}	{}	{}	{}
pre_connection_tests	{}	{}	{}	{}	{}	{}
traceroute	✓	✗	✓	✓	✓	✓

Abbildung 30: Auszug der NAPALM Supportmatrix



Wie man in der napalm-Supportmatrix sieht, werden viele in unserer Arbeit nützliche Funktionen „out of the box“ unterstützt. Sollte eine Funktion fehlen, ist es möglich neue Funktionen selber zu schreiben und hinzuzufügen.

Funktionsweise

Hier der Auszug eines `get_interfaces` calls:

```
{
  "Ethernet2": {
    "is_enabled": true,
    "description": "",
    "last_flapped": 1502731278.4344141,
    "is_up": true,
    "mac_address": "08:00:27:3D:83:34",
    "speed": 0
  },
  "Management1": {
    "is_enabled": true,
    "description": "",
    "last_flapped": 1502731294.598835,
    "is_up": true,
    "mac_address": "08:00:27:7D:44:C1",
    "speed": 1000
  },
  "Ethernet1": {
    "is_enabled": true,
    "description": "",
    "last_flapped": 1502731278.4342606,
    "is_up": true,
    "mac_address": "08:00:27:E6:4C:E9",
    "speed": 0
  }
}
```

Abbildung 31: `get_interfaces` call

Die Ergebnisse des Calls werden als JSON-Objekt geliefert, sodass diese ohne Probleme im Code weiterverarbeitet werden können.

3.2.3 NAPALM-Logs

Da Syslog als Sensor im Netz eingesetzt wird, ist eine offene Frage, wie die plaintext Nachrichten von Syslog in eine im Code nutzbare Form gebracht werden können. Diese Lücke füllt napalm-logs [8].

Napalm-logs ist eine Open Source Software, die als Daemon läuft und auf Syslog Meldungen horcht und diese in eine herstellerunabhängige Form bringt. Anders als napalm, wird napalm-logs nur benutzt, um Nachrichten zu empfangen.

Die Nachrichten werden entweder direkt durch TCP/UDP empfangen oder im Zusammenhang mit einem Messaging-System, wie ZeroMQ oder Kafka, entgegengenommen und verschickt.



Funktionsweise:

Folgende zwei Abbildungen stellen zwei Syslog-Meldungen von zwei unterschiedlichen Geräteherstellern dar:

```
<149>Jun 21 14:03:12 vmx01 rpd[2902]: BGP_PREFIX_THRESH_EXCEEDED: 192.168.140.254 (External AS 4230): Configured maximum prefix-limit threshold(140) exceeded for inet4-unicast nlri: 141 (instance master)
```

Abbildung 34: Syslog Meldung eines Juniper-Devices

```
<149>2647599: xrv01 RP/0/RSP1/CPU0:Mar 28 15:08:30.941 UTC: bgp[1051]: %ROUTING-BGP-5-MAXPFX : No. of IPv4 Unicast prefixes received from 192.168.140.254 has reached 94106, max 12500
```

Abbildung 32: Syslog-Meldung eines Cisco-Devices

Wie man sieht haben beide Nachrichten völlig unterschiedliche Formen, obwohl sie eine Meldung für dasselbe Problem repräsentieren. Napalm-logs nimmt diese Nachrichten und bringt sie in dieselbe Form (siehe rechts).

- **error:** von napalm-logs vordefinierter eindeutiger Identifier (UID)
- **facility & severity:** geben Schweregrad und Ort des Fehlers an (siehe auch Domainanalyse)
- **host:** Hostname des Netzwerkgeräts
- **ip:** IP-Adresse des Geräts
- **os:** Betriebssystem des Geräts
- **timestamp:** UTC timestamp zum Zeitpunkt der Generierung der Nachricht
- **yang_message:** YANG-Modell welches benutzt wurde um die Nachricht zu formen (siehe auch Domainanalyse)

```
{
  "error": "BGP_PREFIX_THRESH_EXCEEDED",
  "facility": 18,
  "host": "vmx01",
  "ip": "127.0.0.1",
  "os": "junos",
  "severity": 5,
  "timestamp": 1498053792,
  "yang_message": {
    "bgp": {
      "neighbors": {
        "neighbor": {
          "192.168.140.254": {
            "afi_safis": {
              "afi_safi": {
                "inet4": {
                  "ipv4_unicast": {
                    "prefix_limit": {
                      "state": {
                        "max_prefixes": 140
                      }
                    }
                  },
                  "state": {
                    "prefixes": {
                      "received": 141
                    }
                  }
                }
              }
            }
          },
          "state": {
            "peer_as": "4230"
          }
        }
      }
    }
  },
  "yang_model": "openconfig-bgp"
}
```

Abbildung 33: Resultierende Meldung



3.2.4 Apache Kafka

Apache Kafka ist eine verteilte *Streaming-Platform* [7]. Eine Streaming-Platform hat folgende Eigenschaften:

- Publish/Subscribe-Mechanismus an einen Stream von *Records* (Ein Record ist eine Key/Value Dateninstanz, welche über einen Zeitstempel verfügt).
- Fehlertolerantes Speichern von Record Streams
- Ermöglicht das Verarbeiten von Daten in Echtzeit

Kafka wird generell für folgende Arten von Anwendungen verwendet:

- Das Aufbauen von Echtzeit-Daten-Pipelines, welche die Daten zuverlässig zwischen Systemen und Anwendungen übertragen.
- Das Aufbauen von Applikationen, welche in Echtzeit auf Streaming-Daten reagieren.

Kafka kann dabei als Cluster von einem oder mehreren Servern laufen und speichert Record Streams als sogenannte *Topics*.



Abbildung 35: Apache Kafka

Kafka API

Kafka verfügt über vier Kern-APIs:

Producer API:

Die Producer API erlaubt es einer Applikation einen Record Stream, zu einem oder mehreren Topics, zu veröffentlichen.

Consumer API:

Die Consumer API erlaubt es einer Applikation, ein oder mehrerer, solcher Topics zu abonnieren.

Streams API:

Die Streams API erlaubt es einer Applikation Streams zu verarbeiten, d.h. den Output eines Topics zu konsumieren und dann einen neuen Output-Stream, zu einem neuen Topic, zu generieren. So ist es möglich, Input-Streams in Output-Streams zu transformieren.

Connector API:

Die Connector API ermöglicht das Aufbauen von wiederverwendbaren Produzenten und Konsumenten, welche Kafka-Topics mit existierenden Anwendungen und Datensystemen verbinden. So kann ein Connector z.B. jede Änderung in einer Datenbank erfassen und mittels Kafka übermitteln.

Die Kommunikation in Kafka zwischen den Clients und dem Server ist TCP-basiert. Es existieren Clients für Kafka-Consumer und -Producer in sowohl Java, wie auch in Python. Kafka Streams ist jedoch nur in Java offiziell unterstützt.

Für die Benutzung in OATs werden nur die Producer-, Consumer-, und Streams-API von Kafka verwendet.

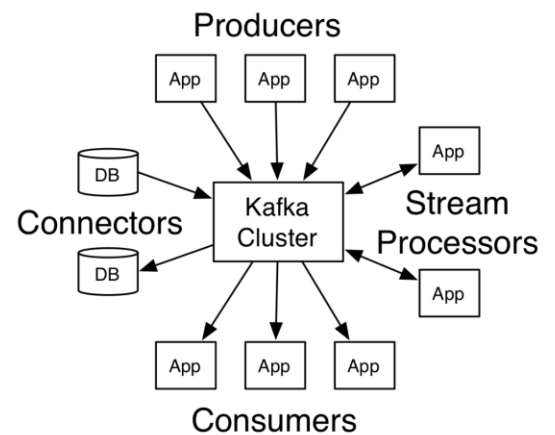


Abbildung 36: Kafka API

Kafka Topics

Ein Topic ist ein Name für eine Kategorie oder einen Feed, zu welchem ein Record veröffentlicht wird. Topics sind bei Kafka immer für mehrere Abonnenten ausgelegt, d.h. ein Topic kann über keinen, einen oder mehrere Konsumenten verfügen.

Für jedes Topic unterhält Kafka eine oder mehrere Partitionen. Dies sieht wie folgt aus:

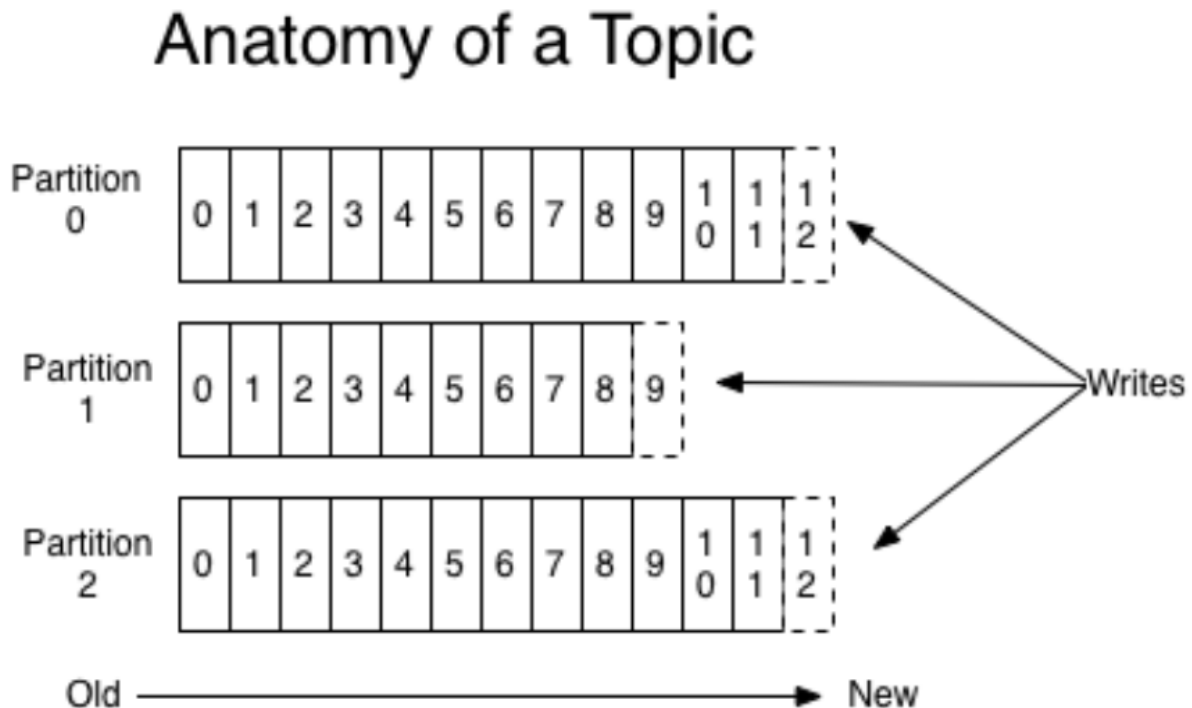


Abbildung 37: Kafka Topic-Partitionen

Jede Partition ist eine geordnete und unveränderliche Sequenz von Records. Jeder Record, innerhalb einer Partition, verfügt über eine ID, welche jeden Record, innerhalb dieser Partition, eindeutig identifiziert.

Kafka persistiert alle Records, unabhängig davon ob sie konsumiert wurden oder nicht, für eine konfigurierbare Zeitdauer. Stellt man die Zeitdauer auf z.B. 24 Stunden ein, so werden alle Records für 24 Stunden gespeichert und danach gelöscht.

3.2.5 Datenanalyse

Um die Use Cases der Event-Korrelation und der Dateiverarbeitung der Streaming Daten zu erfüllen, wird ein Tool bzw. ein Client benötigt, welcher die Daten entgegennimmt und in Echtzeit analysieren kann. Er muss auch in der Lage sein, Vergleiche mit historischen Daten durchzuführen. Zu diesem Zweck werden hier einige Möglichkeiten verglichen, wie diese Use Cases erfüllt werden können.

Hadoop

Hadoop ist ein auf Java basierendes Open Source Framework, mit welchem sich grosse Datenmengen auf verteilten Systemen, in hoher Geschwindigkeit verarbeiten lassen [43].

Hadoop besteht aus folgenden 4 Komponenten:

- Hadoop Common:
Stellt Libraries und Utilities zur Verfügung, welche von den anderen Hadoop Komponenten genutzt werden.
- HDFS (Hadoop Distributed File System)



Java basiertes Datenspeicherungs-System, welches Daten über mehrere Geräte hinweg verteilt speichert.

- Hadoop MapReduce
Java-Framework, mit welcher das Verarbeiten und Analysieren von Daten, über mehrere Geräte hinweg, verteilt werden kann.
- YARN (Yet Another Resource Negotiator)
Ist zuständig für die verschiedenen Prozesse der Hadoop Komponenten

HDFS

HDFS ist ein verteiltes, skalierbares und tragbares Dateisystem [44]. Der Datenzugriff erfolgt über Shell-Commands oder eine Java-Programming-API. Das Dateisystem ist in der Lage, grosse Dateien (>1GB) über mehrere Maschinen zu verteilen. Dabei verteilt es die Dateien nicht nur, sondern repliziert diese auch und erreicht so eine hohe Zuverlässigkeit.

Innerhalb von OATS würde HDFS die Datenbank ersetzen, wenn Hadoop eingesetzt werden soll.

MapReduce

Die MapReduce Engine wird benutzt, um die Daten welche im HDFS abgelegt sind, zu verarbeiten und zu analysieren. Dabei wurde auch hier ein verteilter Ansatz umgesetzt. Die Engine weiss Bescheid, welche Geräte im Netz verfügbar sind und wo die benötigten Daten für einen Analyse-Job liegen. Ein Analyse-Job könnte z.B. das Berechnen des durchschnittlichen Jitters im Netz in der letzten Woche sein. Es ist somit in der Lage, die Jobs effizient auf die Maschinen zu verteilen, sodass so wenig Traffic wie möglich generiert wird. Gleichzeitig stellt die Engine auch sicher, dass bei einem unerwarteten Abbruch eines Jobs, der Job trotzdem auf einer anderen Maschine durchgeführt wird.

Vor- und Nachteile

Vorteile:

- Praktisch unendlich skalierfähig
- Kann mit jeglicher Art von Daten umgehen (müssen nicht strukturiert sein)
- Die MapReduce Engine bietet Funktionen zum Verarbeiten der Daten an
- Ist weiterverbreitet und damit gut dokumentiert
- Kosten können gespart werden, wenn bereits viele Geräte verfügbar sind, auf welchen die Daten gespeichert werden können.

Nachteile:

- Grosses Framework, welches viele Funktionen anbietet, welche nicht direkt in OATS benötigt werden.
- Dementsprechend grosser Aufwand zum Aufsetzen [45].
- Würde das geplante Datenbank-Konzept komplett ändern.



Splunk

Splunk [46] ist ein kostenpflichtiges Log-Analysetool und überwacht und analysiert Maschinendaten aus beliebigen Quellen. Es bietet dabei unter anderem folgende Funktionen an:

- Metriken: speichern von numerischen Datenpunkten, welche nach einer Vielzahl von Faktoren durchsucht werden können.
- Machine Learning: Splunk bietet ein integriertes Machine Learning Toolkit an, mit welchem vorgefertigte Algorithmen auf die Daten angewendet werden können. So können Muster innerhalb der eigenen Daten erkannt werden.
- Event-Sammlung: Sammeln und Korrelieren von Events
- Uvm.

Splunk wird hier nur exemplarisch und kurz vorgestellt, da es kostenpflichtig ist und somit unsere Open Source Anforderungen nicht erfüllt. Es zeigt aber auf, dass schon sehr mächtige Tools auf dem Markt existieren, welche genau anbieten, was OATS von einem Datenanalyse-Tool erwartet. Dies ist einer der Gründe, weshalb die Tools austauschbar sein sollen, damit OATS angepasst werden kann, sollte eine solche Lösung bereits existieren.

Pandas

Pandas [47] ist eine Python Open Source Library welche Datenstrukturen und Datenanalyse-Tools bietet und darauf ausgelegt ist, mit relationalen Daten zu arbeiten.

Die Funktionalitäten die uns im Kontext OATS am meisten interessieren sind:

- Verarbeiten von tabellenartige Daten wie z.B. SQL-Tabellen oder Excel-Sheets
- Verarbeiten von geordneten Timeseries-Daten in Echtzeit

Unsere historischen Daten sind in einer SQL-Datenbank abgelegt, während unsere Timeseries Daten kontinuierlich durch Streaming-Telemetry geliefert werden. Wir wollen in der Lage sein, beide zu verarbeiten. Pandas ist sehr mächtig, wenn es darum geht auf grosse Datenmengen über Datenbankzugriffe zuzugreifen und diese zu manipulieren. Leider ist es nicht dafür ausgelegt, Daten in Echtzeit zu verarbeiten, weswegen es hier zu Problemen beim Umgang mit Streaming-Daten kommen könnte.



Kafka Streams

Kafka ermöglicht das Verarbeiten von Streaming-Daten (*stream processing*) durch die Kafka Streams Library. Der Fokus liegt dabei auf einem einfachen Java-basierten Deployment. Es ist nicht nötig, noch zusätzliche Infrastruktur aufzusetzen wie z.B. bei Hadoop und ist darauf ausgelegt als Microservice (= unabhängiger Prozess) zu laufen.

Die Vorteile von Kafka Streams sind [48]:

- **Leistungsfähigkeit:** Ermöglicht das Entwickeln von skalierfähigen, verteilten und fehlertoleranten Stream-Processing-Anwendung
- **Leichtgewichtigkeit:** Benötigt keine zusätzliche Infrastruktur und hat keine externen Abhängigkeiten ausser Kafka.
- **Echtzeit:** Ermöglicht das Verarbeiten von Daten mit einer sehr geringen Latenz (0.001s<)
- **Sicherheit:** Ermöglicht das Verschlüsseln der Daten und unterstützt Authentisierung und Autorisierung der Clients.

Der grösste Nachteil im Kontext von OATS ist, dass Kafka Streams zurzeit nur als Java-Library zur Verfügung steht (die restlichen Kafka-Libraries stehen auch in Python zur Verfügung).

Hier ein Beispiel einer solche Kafka Streams Java-Applikation (die Import-Statements wurden weggelassen):

```
public class WordCountApplication {

    public static void main(final String[] args) throws Exception {
        Properties config = new Properties();
        config.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount-application");
        config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092");
        config.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
        config.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());

        StreamsBuilder builder = new StreamsBuilder();
        KStream<String, String> textLines = builder.stream("TextLinesTopic");
        KTable<String, Long> wordCounts = textLines
            .flatMapValues(textLine -> Arrays.asList(textLine.toLowerCase().split("\\W+")))
            .groupBy((key, word) -> word)
            .count(Materialized.<String, Long, KeyValueStore<Bytes, byte[]>>as("counts-store"));
        wordCounts.toStream().to("WordsWithCountsTopic", Produced.with(Serdes.String(), Serdes.Long()));

        KafkaStreams streams = new KafkaStreams(builder.build(), config);
        streams.start();
    }
}
```

Abbildung 38: Kafka Streams Java-Applikation

Hier wird der Inhalt vom Topic «TextLinesTopic» gelesen und die Anzahl des Auftretens eines Worts in diesem Topic gezählt. Das aktuelle Wort und die bisherige Anzahl des Auftretens, wird dann an das «WordsWithCountsTopic»-Topic weitergegeben.



Java-Client vs Python-Client

Das OATS-Projekt wurde in der Studienarbeit nur mit Python entwickelt. Leider existieren für ein leichtgewichtiges Stream-Processing zu diesem Zeitpunkt keine Python-Packages, welche als genügend empfunden werden.

Winton-Kafka-Streams [49] ist das einzige Projekt, welches sich anmutet, die gleiche Funktionalität wie Kafka Streams anzubieten. Dies ist leider nur sehr schlecht dokumentiert und zum Zeitpunkt des Verfassens dieses Dokuments ist auch der Build im «Failed»-Status. Nun muss entschieden werden, ob Kafka Streams, welches nur in Java bzw. Scala verfügbar ist, trotzdem als Alternative aufgenommen werden soll.

Die Stream-Processing Clients haben die Aufgabe, von Topics zu lesen und Berechnungen auf den in den Records enthaltenen Daten durchzuführen und die Resultate der Berechnungen weiterzugeben. Es ist irrelevant, ob der Client, welcher diese Aktionen durchführt in Python oder in Java geschrieben wurde, solange die Daten in den Records korrekt deserialisiert und wieder serialisiert werden.

Da das Projektteam Java programmieren kann, wird beim Entscheid für oder gegen Kafka Streams keine Rücksicht auf die Programmiersprache genommen.

Fazit

Hadoop wäre eigentlich ein sehr spannendes Thema, das auch in der realen Welt immer mehr verbreitet ist. Leider ist der Aufwand verglichen mit dem Nutzen, den man durch das ganze Hadoop-Ökosystem hat, im Kontext von OATS nicht ausreichend, um sich für den Einsatz von Hadoop zu entscheiden.

Splunk wäre als All-in-One Lösung eigentlich das perfekte Tool, ist aber leider nicht Open-Source, weswegen auch das wegfällt.

Pandas ist eine mächtige Python-Library für den Umgang mit Daten. Leider wurde es aber nicht dafür ausgelegt, in Echtzeit Daten zu verarbeiten. Da OATS gut skalieren soll, ist der Umgang mit Streaming-Daten essentiell und wir wollen nicht riskieren, dass es hier zu Problemen kommt.

Deswegen haben wir uns gegen Pandas entschieden.

Damit bleibt noch Kafka Streams übrig, welches inhärent dafür ausgelegt wurde, Streaming-Daten von Kafka zu verarbeiten. Wichtig ist hierbei auch, dass Kafka-Streams, keine nicht benötigten Bestandteile mit sich zieht wie z.B. Hadoop. Der einzige Nachteil ist im Moment die Programmiersprache Java, da die entwickelten Clients aber innerhalb von Kafka gut austauschbar sind, sollte dies kein Problem darstellen. Sollte zu einem späteren Zeitpunkt eine brauchbare Kafka Streams Python-Library verfügbar sein, können weitere Clients einfach in Python entwickelt werden.

Fazit: Wir entscheiden uns dafür, für die Datenanalyse Kafka Streams zu verwenden.



3.2.6 vflow

Da die Netflow-Daten nicht wie geplant über Streaming-Telemetry übermittelt werden können, wird ein Client benötigt, welcher die Netflow-Daten entgegennimmt und sie in OATS weiterleitet. Zu diesem Zweck wurde vflow [10] gefunden.

Vflow fungiert als leichtgewichtiger Service, welcher eine IP und Portnummer angibt, welche als Netflow-Collector dient. Dann werden auf den Netzwerkgeräten Flow-Exporter konfiguriert, welche diese IP/Portnummer als Endpunkt haben und somit alle Flows dorthin versenden.

Der weitere Transport der Daten funktioniert dann standardmässig über Kafka.

Vflow ist somit das Netflow-Pendant zu napalm-logs, nur dass es komplett *stateless* ist, da die Daten nicht transformiert werden müssen.

3.2.7 Troubleshooting- und Konfigurationsmanagement-Tools

Diese Analyse wurde bereits als Teil der Studienarbeit durchgeführt.

Für den Core unserer Software haben wir drei Tools genauer betrachtet und diese verglichen, damit wir schlussendlich dasjenige verwenden, welches unseren Anforderungen am besten entspricht. Die Tools sollen in der Lage sein, erhaltene Events zu interpretieren und weiterzuverarbeiten, basierend auf vordefinierten Regeln [50].

Ansible

Ansible [51] ist das erste Tool, welches wir genauer betrachtet haben. Um Ansible zu verwenden, muss man es bloss auf einem Linux Gerät installieren und man kann loslegen. Dies funktioniert, da Ansible keine Daemons und keine Datenbanken verwendet.

Damit Ansible weiss mit welchen Geräten es kommunizieren soll reicht es aus, ein statisches File mit Adressen zu hinterlegen, man kann aber auch ein dynamisches Discovery Modul, wie etwa Amazon EC2 oder Openstack verwenden, damit man relevante VMs basierend auf API calls finden kann. Sobald man ein Geräteinventar erstellt hat, kann man damit beginnen, Variablen und Parameter zu definieren, welche von Ansible Playbooks verwendet werden können. Der Zugriff auf Ansible selbst läuft über die Kommandozeile.

Playbooks sind Sequenzen von Ansible Modulen, welchen man auf einem Host ausführen möchte, geschrieben sind sie in YAML. Ansible spricht Server über SSH an, da dies bei allen Endgeräten vorhanden ist. Sobald Ansible verbunden ist, wird der Python Code übertragen und ausgeführt, nach dem ausführen wird er wieder gelöscht.



ANSIBLE

Abbildung 39: Ansible

Architektur

Die Ansible-Applikation läuft auf dem Gerät, auf welchem man es installiert hat, falls etwas auf einem anderen Host ausgeführt werden muss, wird eine Verbindung erstellt und nach dem Ausführen wieder beendet. Es existiert also keine Server-Client Architektur, sondern man muss Aufgaben auf dem Hauptgerät parallelisieren.

Ansible gibt es auch in der Enterprise „Tower“-Variante, dies erlaubt es einem Ansible auf mehrere Server zu verbinden und somit stark zu skalieren.

Da Ansible nur auf einem Gerät läuft, ist es sehr einfach auf eine neuere Version zu upgraden, ohne dass auf einem entfernten Gerät Kompatibilitätsprobleme auftauchen.



Enterprise Version

Mit der Verwendung der Enterprise Version kann man anstelle der Kommandozeile zusätzlich auch ein Web Interface verwenden. Dies erlaubt es einem, ein Notifikation System anzubinden sowie Aufträge selber zu planen.

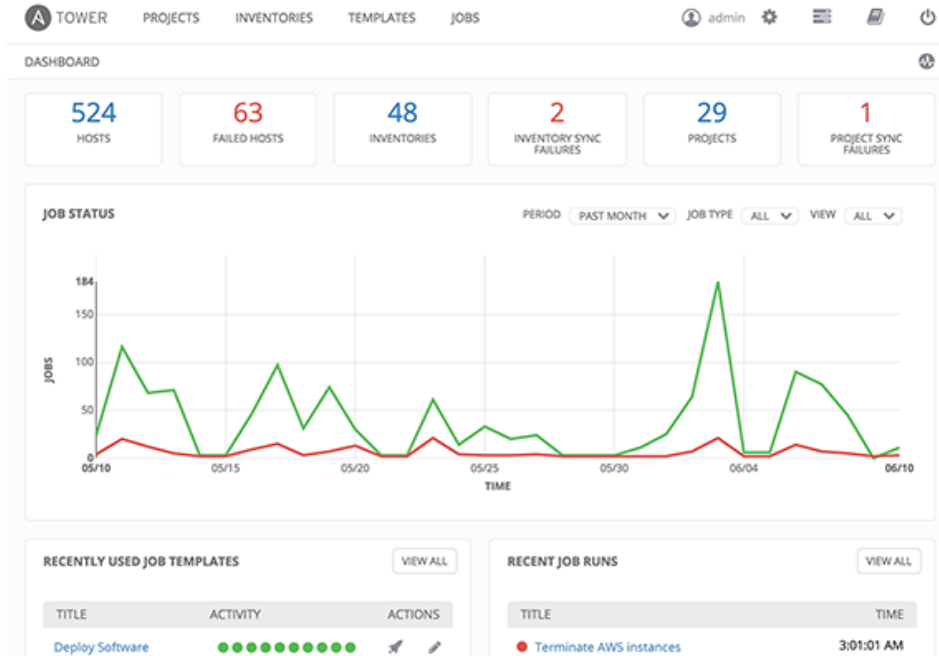


Abbildung 40: Ansible Tower

Mit dem Interface kann man das aktivieren von Playbooks direkt automatisieren und die gewünschten Ziel VMs aus der Inventarliste auswählen. Diese Zusatzfunktionen kommen aber mit einem Preis.

SELF-SUPPORT	STANDARD	PREMIUM
PERFECT FOR SMALL DEPLOYMENTS	FOR ENTERPRISE IT OPERATIONS	FOR MISSION-CRITICAL DEVOPS
Support not included Maintenance & upgrades Limited features	8x5 support Maintenance & upgrades Enterprise features	24x7 support Maintenance & upgrades Enterprise features
Up to 100 nodes \$5,000/year	Up to 100 nodes \$10,000/year \$13,000/year with Ansible Engine	Up to 100 nodes \$14,000/year \$17,500/year with Ansible Engine Ansible Networking support available
Up to 250 nodes \$10,000/year	Over 100 nodes Contact us for pricing	Over 100 nodes Contact us for pricing
Over 250 nodes Standard & Premium only	View production support service level	View production support service level

Abbildung 41: Ansible Support



Ansible Support

Ansible wird von allen wichtigsten Netzwerkgeräthändlern und Plattformen unterstützt. Ansible erlaubt es einem, durch das Ausführen von plattformspezifischen Modulen und Playbooks, die Konfiguration eines Netzwerk-Stacks, von einem System aus zu automatisieren. Es erlaubt einem ausserdem, das existierende Netzwerk zu testen und zu validieren, der Informationsgewinnungsprozess kann selbst implementiert werden, um Informationen über eine existierende Konfiguration zu gewinnen.

Ansible unterstützt die Programmierbaren Cisco Plattformen, Arista, Juniper aber auch andere Anbieter. Die Unterstützung kommt von den Händlern selbst und nicht von der Community.

Vor- und Nachteile

Vorteile:

- Schneller und einfacher Start durch minimalen Installationsprozess
- Verschiedene Dokumentationen und Module sind von der Community bereits vorhanden
- Netzwerk Module von den Anbietern selbst verfügbar

Nachteile:

- Kein System um Events zu triggern oder auf diese zu reagieren
- Viele Funktionen nur in der Premium Version verfügbar



Stackstorm

Stackstorm [52] ist eine erweiterbare Regel- und Ausführungs-Engine, die diverse Konfigurationen und Einstellungen erlaubt.

Dies ermöglicht es, eine sehr flexible, auf die jeweiligen Anforderungen angepasste, Lösung zu erstellen. Man kann Regeln

erstellen, welche es erlauben auf spezifische Events zu reagieren und Befehle oder komplexe Workflows zu deployen. Stackstorm bietet ausserdem einen Service für Chat-Ops, was es erlaubt mit Chat Plattformen wie etwa Slack direkt auf Funktionen zuzugreifen.



Abbildung 42: Stackstorm

Architektur

Die Architektur von Stackstorm besteht aus sechs Hauptteilen, welche hier kurz beschrieben werden.

Sensoren sind Python-Plugins welche Events erhalten oder aktiv nach Events Ausschau halten. Falls ein Event in einem externen System vorkommt, werden diese von den Stackstorm-Sensoren registriert und ein Trigger wird ins System eingegeben.

Ein Trigger ist die Repräsentation eines externen Events, es gibt zwei Kategorien, generische Trigger wie etwa ein ablaufender Timer und integrierte Trigger, wie etwa ein Monitoring-Alarm oder ein JIRA Issue Update. Neue Trigger können über die Sensoren definiert werden.

Sobald ein Trigger im System erscheint wird eine Action ausgeführt, hier gibt es drei Kategorien, generische Actions, wie ein SSH Aufruf oder ein REST call, die integrierten Actions wie Openstack oder Puppet und schliesslich die komplett benutzerdefinierten Actions. Hier werden ebenfalls Python-Plugins oder einfache Scripts verwendet. Actions können manuell über die Kommandozeile oder die API gestartet werden oder basierend auf einer Regel.

Rules sind dazu da die vorkommenden Trigger einer Aktion oder einem Workflow zuzuordnen, basierend auf erfüllten Kriterien, wie Payload oder Tags.

Workflows sind zusammengesetzte Aktionen in komplexe Operationen, hier werden Reihenfolge, Kriterien und Datenübertragungen definiert. Oft gibt es für Automations-Operationen mehrere Schritte und dementsprechend auch mehrere Aktionen. Ein Workflow kann, wie eine einfache Aktion, aus der Bibliothek manuell gestartet werden oder sie werden durch eine Regel gestartet. Packs können sowohl Trigger und Aktionen aber auch Regeln und Workflows beinhalten, sie erlauben ein einfaches Managen und Teilen von benutzerdefinierten Stackstorm-Inhalten. Die Anzahl an Benutzerdefinierten Packs, welche über die Stackstorm Community erhältlich sind wächst stetig. Ausgeführte Aktionen können, manuell oder automatisiert, in einem sogenannten Audit-Trail gespeichert werden. Dort werden die kompletten Daten einer Aktion, über Kontext und Resultat, erfasst.

Stackstorm verwendet eine Anzahl von verschiedenen Services, wie etwa ein Rabbit-Message-Queue und eine MongoDB zur Datenspeicherung.

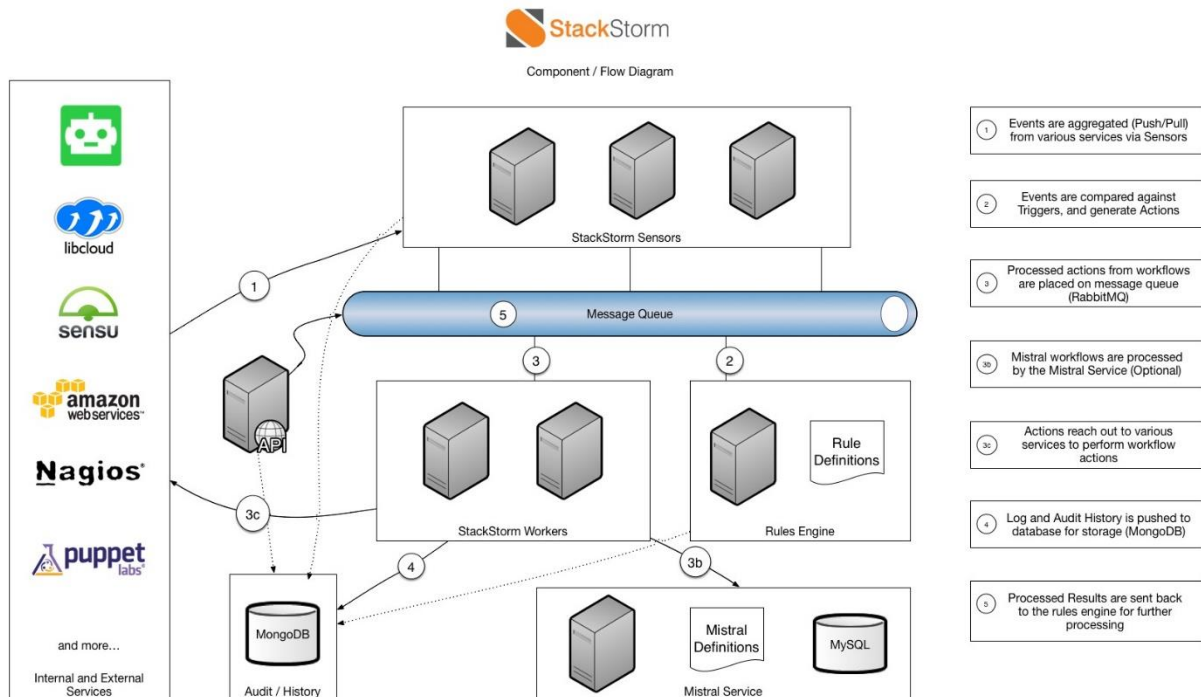


Abbildung 43: Stackstorm Infrakstruktur

Stackstorm wurde nicht dazu erstellt, um Endpunkte zu konfigurieren oder mit ihnen zu kommunizieren, dazu bietet es Pakete für Salt, Puppet oder Ansible an. Dies erlaubt es Funktionen und Vorteile von anderen Plattformen zu nutzen und mit Stackstorm zu kombinieren. Logs und die Audit-History werden über MongoDB in einer skalierbaren Umgebung abgelegt. Die angebotenen Services sind als Http- oder Restful-API konzipiert, man kann einen einzelnen Server oder mehrere verwenden, je nach Anforderungen.

Stackstorm gibt es in seiner Originalfassung ohne Erweiterungen, es muss also alles zusätzlich installiert werden. Dies macht Stackstorm an sich schlanker und die Abhängigkeiten minimal. Wenn mittels Stackstorm entwickelt wird, kann man selbst erstellte Pakete zusammen in eine einzelne Version bündeln. Jedes Paket von Stackstorm hat seine eigenen Requirements-Files, es erlaubt ausserdem beim Upgrade von Paketen die Konfigurationen zu behalten.

Die Dokumentation liegt ausserhalb des eigentlichen Modul-Codes, was beim Einsatz von fremdentwickelten Paketen zu Schwierigkeiten führen kann, da diese eventuell nicht sauber beschrieben sind.

Stackstorm ist ein Apache-2 lizenziertes Open Source Software Produkt, gehostet wird es auf Github

Stackstorm Support

Brocade VLX und SDX bieten sehr guten Support für Stackstorm, durch ein verbinden des Supports mit der napalm library, bieten sie auch Support für Cisco, Juniper, Arista und weitere Anbieter.

Vor- und Nachteile

Vorteile:

- Gratis Web-Interface ist einfach zu bedienen, auch mit sehr wenig Python- oder Programmierkenntnissen.
- Direkte Einbindung von Chat-Ops, Unterstützt diverse Chat Plattformen wie etwa Slack
- Sehr mächtige Regel Engine

Nachteile:



- Weniger Extensions im Vergleich zu anderen Tools, Systeme und API werden eventuell nicht unterstützt
- Workflows sind oft «trial and error», da der YAQL query syntax nicht gut dokumentiert ist
- Stackstorm benötigt zur Kommunikation mit Geräten noch weitere Plugins



Salt

Im Gegensatz zu den beiden anderen Tools wurde Salt [11] (das Open Source Produkt von Saltstack) als verteiltes System konzipiert. Mittels sogenannten «minions» werden auf (Remote-)Objekten Befehle und Abfragen durchgeführt. Die minions können dabei einzeln oder aufgrund von Selektionskriterien angesprochen werden.



Salt ist in der Lage Remote-Objekte direkt zu konfigurieren. Dies passiert mittels sogenannten «states». Mittels den states wird sichergestellt, dass bestimmte Pakete oder Services installiert und im Betrieb sind.

Abbildung 44: Saltstack

Salt besteht unter anderem aus folgenden Komponenten:

- **master:** Der Server, auf dem die Corefunktionalität ausgeführt wird und mit den minions kommuniziert.
- **minions:** Objekte, die eine verkleinerte Version von Salt ausführen um die lokale Ausführung von Befehlen und Kommunikation mit dem master zu ermöglichen. Minions werden mittels «grains» oder «pillars» angesprochen, die weiter unten erklärt werden.

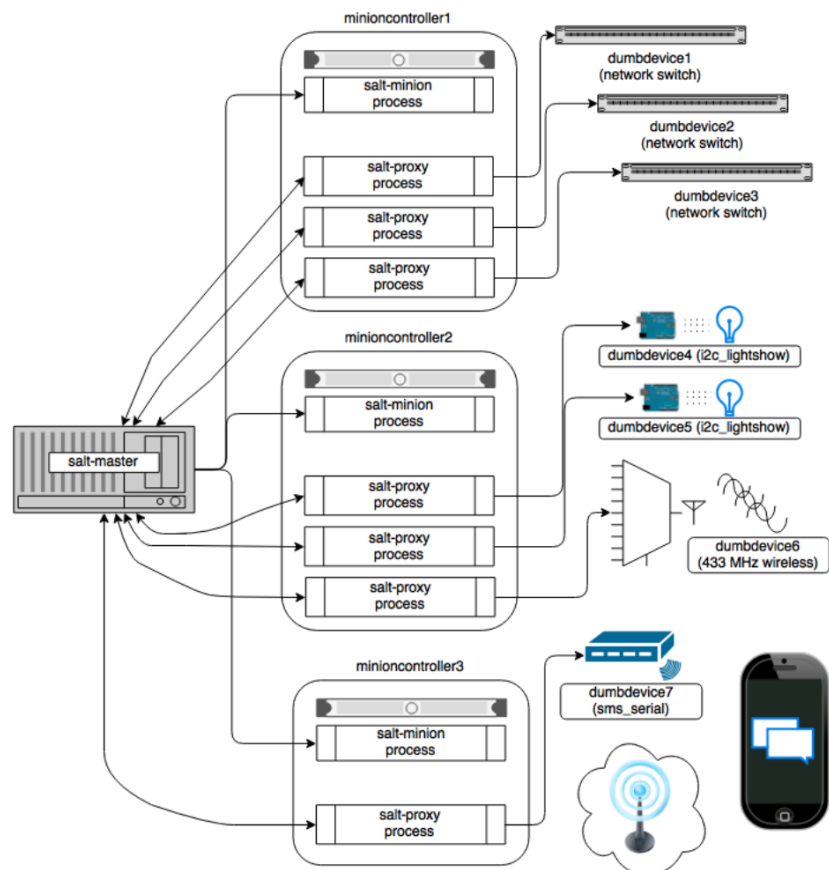


Abbildung 45: Salt Funktionsweise

- **engines:** Salt-Engines sind extern laufende Prozesse, auf denen z.B. auf Events gehorcht werden kann.
- **states / formulas:** Dateien die YAML oder Templating-Daten enthalten um die minions zu konfigurieren.
- **grains:** Grains bieten ein Interface um «Körner» von Informationen von Geräten zu sammeln (grain = Korn). So können Informationen über das OS, IP-Adresse, domain name etc. des darunterliegenden Systems gewonnen werden.
- **pillars:** Pillars bieten ein globales Interface, um Werte an die minions zu verteilen. Sie können in mehreren Formaten gespeichert werden (JSON, YAML und andere) und können lokal oder remote liegen.
- **mine:** Die Salt-mine sammelt Daten der minions und speichert sie auf dem master. Der Unterschied zu den grains besteht darin, dass die Daten aktueller sind, da grains nicht oft

aktualisiert werden. Die mines können so, in einem regelmässigen Intervall, Daten von den Minions abfragen, um einen aktuellen Status über das gesamte Netzwerk zu erlangen.

Architektur

Salt liegt eine «hub and spoke»-Architektur zugrunde. Es wäre auch ein Deployment mit mehreren masters möglich. Ein master kann aber auf mehrere Tausend Knoten skaliert werden, sodass ein einzelner master in den meisten Fällen ausreichend ist.

Der master enthält state-Dateien, welche als Baumstruktur aufgesetzt sind. So können einfach verschiedene Servergruppen angesprochen und konfiguriert werden.

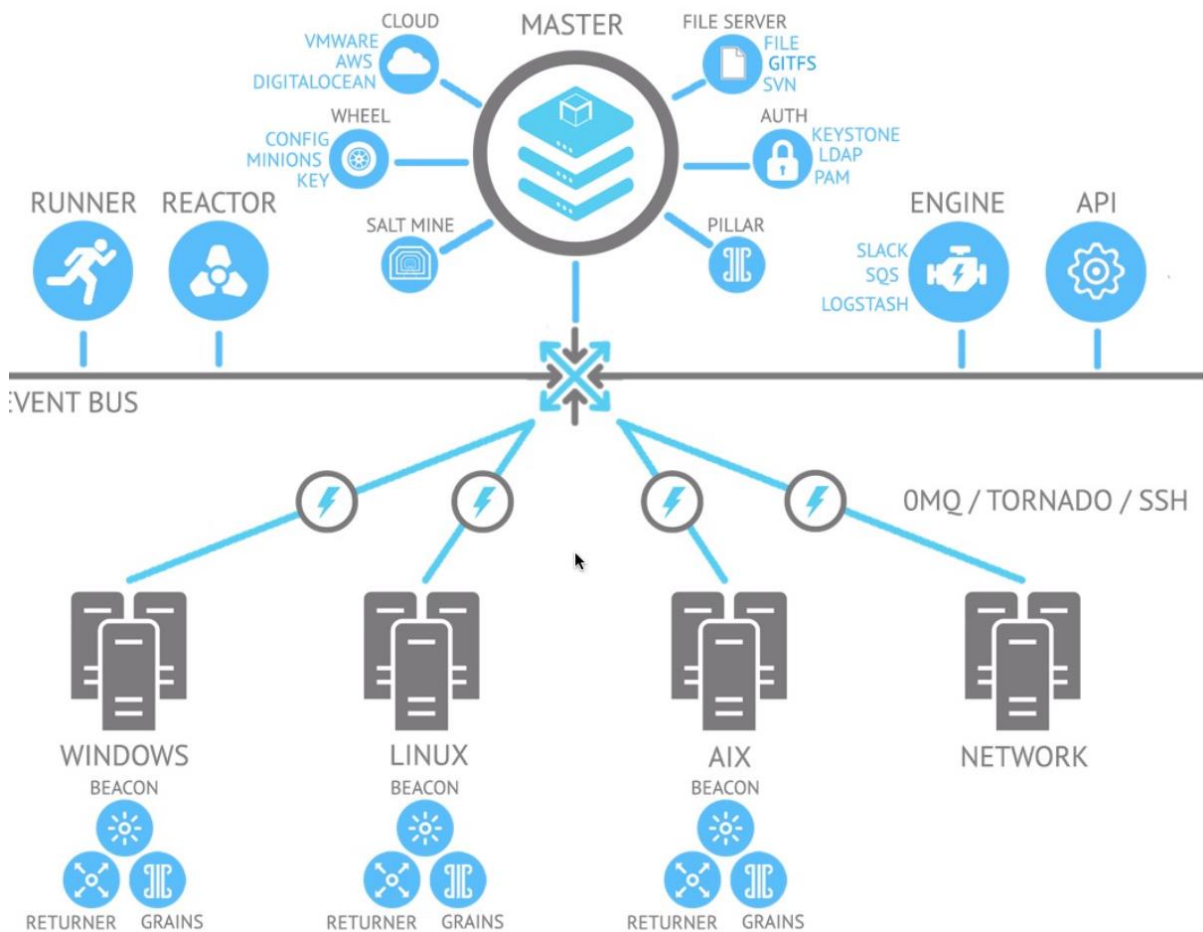


Abbildung 46: Salt Architektur

Das Event-basierte System von Salt benutzt «beacons», ähnlich zum Sensor- und Trigger-System von Stackstorm. Die beacons leiten Events in den Nachrichtenbus, welche anschliessend vom «reactor» auf dem master verarbeitet werden. Der reactor enthält eine Regel-Engine, mit welcher ein state oder ein direkter Befehl ausgeführt werden kann. Da die beacons auf den minions laufen, ist die Übermittlung von Events an den master sehr einfach. Nur Salt ist in der Lage dies so zu tun, da Ansible und Stackstorm agentless (→keine minions) sind.

Die Regel-Engine vom reactor ist verglichen mit demjenigen von Stackstorm eher begrenzt. Der «thorium complex reactor» ist hingegen in der Lage, komplexere Abläufe zu definieren.

Erweiterbarkeit

Dies ist eine der Stärken von Salt. Der einfachste Weg zum Erweitern von Salt ist es, neue state(wie ein Objekt konfiguriert werden soll) oder execution-(Code um mit einem System zu interagieren) modules zu schreiben.



Salt-Packs sind Pakete, die in Salt integriert werden können. So existieren z.B. Packs für napalm und napalm-logs (und viele andere). Auch können eigene Packs entwickelt und veröffentlicht werden.

Salt Support

Zurzeit werden folgende Netzwerk-Betriebssysteme unterstützt:

- JunOS (Juniper)
- NXOS (Cisco)
- Cisco NSO (Cisco NETCONF)
- NAPALM

Dadurch, dass napalm unterstützt wird, werden indirekt auch alle Funktionen unterstützt, die napalm anbietet. Dies erweitert den Funktionsumfang beträchtlich.

Vor- und Nachteile

Vorteile:

- Ermöglicht agent- und auch agentless-Deployment
- Sehr gut dokumentiert (inkl. einfacher Tutorials mit vorbereiteter Testumgebung)
- Einfache Event-Übermittlung dank der verteilten minions
- Die Regel-Engine ermöglicht die Ausführung von Workflows
- Sehr hohe Performance da als Eventbus ZeroMQ benutzt wird
- Durch die gute Erweiterbarkeit einfach an andere APIs und Datenbanken anhängbar

Nachteile:

- Weniger gute Regel-Engine als Stackstorm (wenn nicht thorium reactor benutzt wird)
- Bei Benutzung von thorium reactor: Noch nicht ausführlich getestet und dokumentiert
- Höherer Verwaltungsaufwand durch die verteilten minions

Fazit

Ansible ist das Tool, mit dem am schnellsten etwas umgesetzt werden könnte. Allerdings ist es das einzige Tool, das keine Event-Driven Architektur unterstützt. Im Grunde genommen ist Ansible nichts anderes als eine Skriptsprache. In der Enterprise-Version wäre Ansible wohl ein wenig mächtiger, doch wir wollen uns im Projekt auf Open Source Lösungen konzentrieren. Deswegen haben wir uns gegen Ansible entschieden.

Die Entscheidung bei Salt vs. Stackstorm ist uns um einiges schwieriger gefallen. In einer perfekten Welt, würden wir uns noch eine Woche Zeit nehmen, um eine Testumgebung aufzusetzen und beide Tools darauf ausgiebig vergleichen. Da uns diese Zeit fehlt, müssen wir zu diesem Zeitpunkt eine Entscheidung treffen.

Für Stackstorm spricht die mächtigere Regel-Engine zum Umsetzen der Workflows. Bei Salt ist die verteilte master – minion Architektur ein klarer Vorteil. Diese würde die Kommunikation zwischen unserem System und den Endgeräten massiv vereinfachen. Auch scheint der thorium reactor in etwa gleich mächtig zu sein, wie die Stackstorm-Engine.

Beide Tools unterstützen napalm, was eine gute API Grundlage für beide Systeme bietet, allerdings ist Salt durch das Modul-System einfacher erweiterbar (eine unserer nichtfunktionalen Anforderungen).

Salt hat im Gegensatz zu Stackstorm kein integriertes Nachrichten System und auch kein GUI (in der Open Source Version). Als Nachrichtensystem kann man mit Salt aber problemlos Slack anhängen.

Um einen besseren Einblick in die beiden Umgebungen zu erhalten, haben wir zusätzlich die Testversionen angeschaut. Auf der Salt Website konnte eine einfache Testumgebung mit einem Master und 2 minions runtergeladen werden. So konnte an einem Beispiel direkt die Funktionsweise ausprobiert werden. Bei Stackstorm wurde zwar auch ein Tutorial angeboten, allerdings ohne Testumgebung, was den Nutzen des Tutorials doch schmälert.



Der Hauptgrund der gegen Stackstorm spricht ist, dass es nicht in der Lage ist Gerätekonfigurationen selbst zu managen. Für diesen Punkt müsste auf ein zusätzliches Tool (Salt, Puppet) zurückgegriffen werden.

	Salt	Stackstorm
Aufbau und Architektur	Master-minion, stark skalierbar, agentenbasiert	Erweiterbare Regeln- und Ausführungs-Engine
Punkte (Max 10)	8	8
Kommunikation zu Endpunkten	Über minions oder proxies	Nicht dafür entworfen, benötigt zusätzliches Tool
Punkte (Max 10)	10	3
Erweiterbarkeit	Erweiterbar durch Module, Abhängigkeiten nicht sauber, sehr umfassend	Erweiterbar durch Packs, sauber individuell installiert, weniger umfassend
Punkte (Max 10)	8	8
Benötigtes Know-How und verfügbare Quellen	Mittlere Länge zum Einrichten viel verfügbare Dokumentation	Mittlere Länge zum Einrichten wenig verfügbare Dokumentation
Punkte (Max 10)	9	6
ChatOps	Muss selbst hinzugefügt werden	Bereits Integriert
Punkte (Max 5)	2	5
Tutorial	Angeboten mit Testumgebung	Angeboten ohne Testumgebung
Punkte (Max 5)	5	2
Benutzeroberfläche	Kein UI	Gratis default WebUI
Punkte (Max 5)	0	5
Total (Max. 55)	42	37

Salt liegt in der Punktzahl leicht vorne, jedoch ist zu beachten, dass wir uns hier auf externe Quellen basieren und unsere eigene Erfahrungen nur auf Artikeln und den kurzen Tutorials basieren. Die Punkte sind zudem unsere eigene Interpretation der verfügbaren Informationen. Unsere Hauptpunkte sind jeweils mit zehn Punkten gewertet, in diesen alleine schliesst Salt deutlich besser ab, vor allem das die Kommunikation zu den Endpunkten direkt stattfinden kann, bei Stackstorm muss hier ein weiteres Tool hinzugefügt werden, was zusätzliche Arbeit und Zeit kosten würde.

In Betrachtung dieser Punkte sind wir zum Schluss gekommen, dass das Herzstück unseres Systems mit Salt umgesetzt werden soll.



3.2.8 ZeroMQ

ZeroMQ wird als Event-Queue innerhalb von Salt benutzt und wird deshalb hier kurz vorgestellt. ZeroMQ ist eine auf hohe Leistung ausgelegte, asynchrone messaging-library für verteilte Software-Systeme und bietet eine Message-queue an. Im Gegensatz zu anderen ähnlichen Systemen, benötigt es keinen message-broker. Die ZeroMQ-API wurde so entworfen, dass sie ähnlich wie Berkeley-Sockets funktioniert (siehe Abbildung 30) [9].

```
import zmq
import time
context = zmq.Context()

subscriber = context.socket (zmq.SUB)
subscriber.connect ("tcp://192.168.55.112:5556")
subscriber.connect ("tcp://192.168.55.201:7721")
subscriber.setsockopt (zmq.SUBSCRIBE, "NASDAQ")

publisher = context.socket (zmq.PUB)
publisher.bind ("ipc://nasdaq-feed")

while True:
    message = subscriber.recv()
    publisher.send (message)
```

Abbildung 47: ZeroMQ Code Beispiel

ZeroMQ unterstützt folgende grundlegende Messaging-Patterns:

- **Request-reply:**
Verbindet ein Set von Publishern mit einem Set von Services
- **Publish-subscribe:**
Verbindet ein Set von Publishern mit einem Set von Subscribern.
- **Push-pull (pipeline)**
Verbindet Knoten in einem *fan-out/fan-in* pattern. Erlaubt mehrere Schritte und auch Loops.
- **Exclusive Pair:**
Verbindet direkt zwei Sockets miteinander.

3.2.9 Slack

Falls ein Problem auftaucht welches nur von einem Techniker vor Ort gelöst werden kann, muss man diesen anbieten. Diese Mitteilungen werden anfangs über Slack gesendet, mit der Option später noch weitere Mitteilungsoptionen anzubieten [53].

Der Name Slack ist ein Akronym für "Searchable Log of All Conversation and Knowledge" und die Software ist im Grunde eine Chat App. Slacks Features beinhalten, persistente Chatrooms auch Channels genannt, welche nach Thema organisiert sind, zusätzlich gibt es auch private Gruppen und Direktnachrichten. Der Inhalt von Slack kann durchsucht werden und beinhaltet Files, Konversationen und Benutzer. Slack bietet zudem eine Integration von diversen Software und Service von Drittanbietern, sowie Support für Community erstellte Integrationen von eigenen Projekten.

Slack ist für uns besonders interessant da es neben dem Mitteilungsdienst zusätzlich noch die Option bietet, einfache Workflows über Nutzereingaben oder Buttons zu starten. Zudem kann man Slack mit einem Modul einfach in eine Salt Installation einbinden.



3.2.10 Datenspeicherung

Netbox: Netzwerk Daten

Für die Bachelorarbeit haben wir uns dazu entschieden, die Speicherung der Netzwerkdaten auf Netbox zu migrieren. Netbox ist eine Open Source Web-Applikation und wurde erstellt um dabei zu helfen, die Verwaltung und Dokumentation eines Netzwerks zu erleichtern. Netbox wurde spezifisch dafür entwickelt, um die Bedürfnisse von Netzwerk- und Infrastruktur-Ingenieuren abzudecken.

Netbox deckt folgende Bereiche ab:

- **IP address management (IPAM)** - IP networks and addresses, VRFs, and VLANs
- **Equipment racks** - Organized by group and site
- **Devices** - Types of devices and where they are installed
- **Connections** - Network, console, and power connections among devices
- **Virtualization** - Virtual machines and clusters
- **Data circuits** - Long-haul communications circuits and providers
- **Secrets** - Encrypted storage of sensitive credentials

Abbildung 48: Daten, welche in Netbox gespeichert werden können

InfluxDB

InfluxDB ist eine Open Source Timeseries-Datenbank. Das heisst, sie ist optimiert dafür Daten zu speichern, welche über das Zeit-Feld indexiert werden. Sie eignet sich also optimal für Operation-Monitoring, Applikationsmetriken und Echtzeit-Analyse von Daten.

InfluxDB hat keine externen Abhängigkeiten und bietet eine SQL ähnliche Sprache zur Verarbeitung der Daten. Die Datenstruktur innerhalb der Datenbank besteht aus Measurements. Ein Measurement setzt sich aus Serien und Punkten zusammen. Ein Punkt ist über mehrere Key-Value Paare und einen Timestamp definiert. Eine Serie wird definiert als Gruppierung von Key-Value Paaren und einem oder mehreren Tags. Diese Serien werden dann über einen String gruppiert, der den Measurement bildet. Die Measurements sind vergleichbar mit den Tabellen aus herkömmlichen Datenbanken.

Weitere Funktionen, welche die InfluxDB bietet, sind die Retention-Policies, welche definieren und kontrollieren wie und wann Daten «downsampled» und eventuell entfernt werden. Downsampling ist die Reduktion einer Zeitreihe, um Speicherplatz zu sparen und ist mit Informationsverlust verbunden.

Troubleshooting Fälle

Wie bereits erwähnt, muss man für Netbox eine PostgreSQL Datenbank aufsetzen, in welcher die Daten für die Netzwerkgeräte gespeichert werden. Da es keinen Sinn macht zwei verschiedene Datenbanken zu verwenden, werden diese Daten ebenfalls in die PostgreSQL-Datenbank migriert.

PostgreSQL

PostgreSQL ist ein Open Source objektrelationales Datenbank Management System.

Features und Funktionen von PostgreSQL sind Multi-Version Concurrency-Control, asynchrone Replikation, nested-transactions, Hot-Backups und ein Query-Planer und -Optimierer. PostgreSQL bietet hohe Skalierbarkeit in Bezug auf Datenmenge und Benutzer. Unter anderem bietet PostgreSQL über ein Framework die Möglichkeit eigene Datentypen zu definieren und zu erstellen, sowie Funktionen und Operationen, um ein passendes Verhalten zu implementieren. Auch gibt es bereits Erweiterungen für diverse Netzwerkadrestypen, was für uns besonders interessant ist, wie etwa ein Datentyp für MAC-Adressen.



3.2.11 Grafana

Im Rahmen der Bachelorarbeit sollen unter anderem auch die vorhandenen Daten visualisiert werden. Dazu wird Grafana verwendet, eine Open Source Applikation.

Mit Grafana kann ein Dashboard entwickelt werden, welches zu verschiedenen Datenquellen abfragen erstellt und die erhaltenen Informationen visualisiert. Beispiele zur Art der Visualisierung sind einfache Graphen, Histogramme, Heatmaps, usw.. Mögliche Datenquellen sind Influxdb oder PostgreSQL-



Abbildung 49: Grafana Dashboard

Das Ziel ist, ein Dashboard zu entwickeln, welches uns schnell einen Überblick über unser System und das Netzwerk liefert.

3.2.12 Flask

Flask ist ein Python basiertes Webframework. Es wird als Micro Framework klassifiziert, da es keine speziellen Tools oder Libraries benötigt. Flask bietet diverse Features an, die für uns interessantesten sind folgende:

- built-in development server and debugger
- integrated unit testing support
- RESTful request dispatching

Mit Flask wollen wir eine API anbieten, die man ansteuern kann, um manuell Events an OATS zu senden. Flask eignet sich besonders gut für diese Aufgabe, da man bereits mit sehr wenig Code eine solide Schnittstelle erstellen kann. Falls man die API und das manuelle absetzen von Probleme noch weiter ausbauen möchte, bietet Flask noch weitere Features und Funktionalitäten an. So bietet sie zum Beispiel Client Side Session über secure Cookies an, falls man nach dem Absetzen eines Problemfalls, noch direkt im Browser, Updates erhalten möchte.

Flasgger

Flasgger ist eine Erweiterung für Flask, mit der man einfach eine API Dokumentation, auch API docs genannt, aufsetzen kann. Dies ist eine graphische Nutzeroberfläche mit welcher man seine API Ressourcen visualisieren kann. Ausser kann man hier auch bereits Nutzerinteraktionen einbinden. Zudem bietet Flasgger auch noch eine Validierung der Daten, die an die API gesendet werden. Die Definition der Datenschemas können über YAML, Python dictionaries oder Marshmallow Schemas erstellt werden.

3.3 Architektur: Stand Studienarbeit / Beginn Bachelorarbeit

Dieses Kapitel dient zum Vergleich der Architekturen zu Beginn und zum Ende der Bachelorarbeit
Am Ende der Studienarbeit sah die Architektur von OATS wie folgt aus:

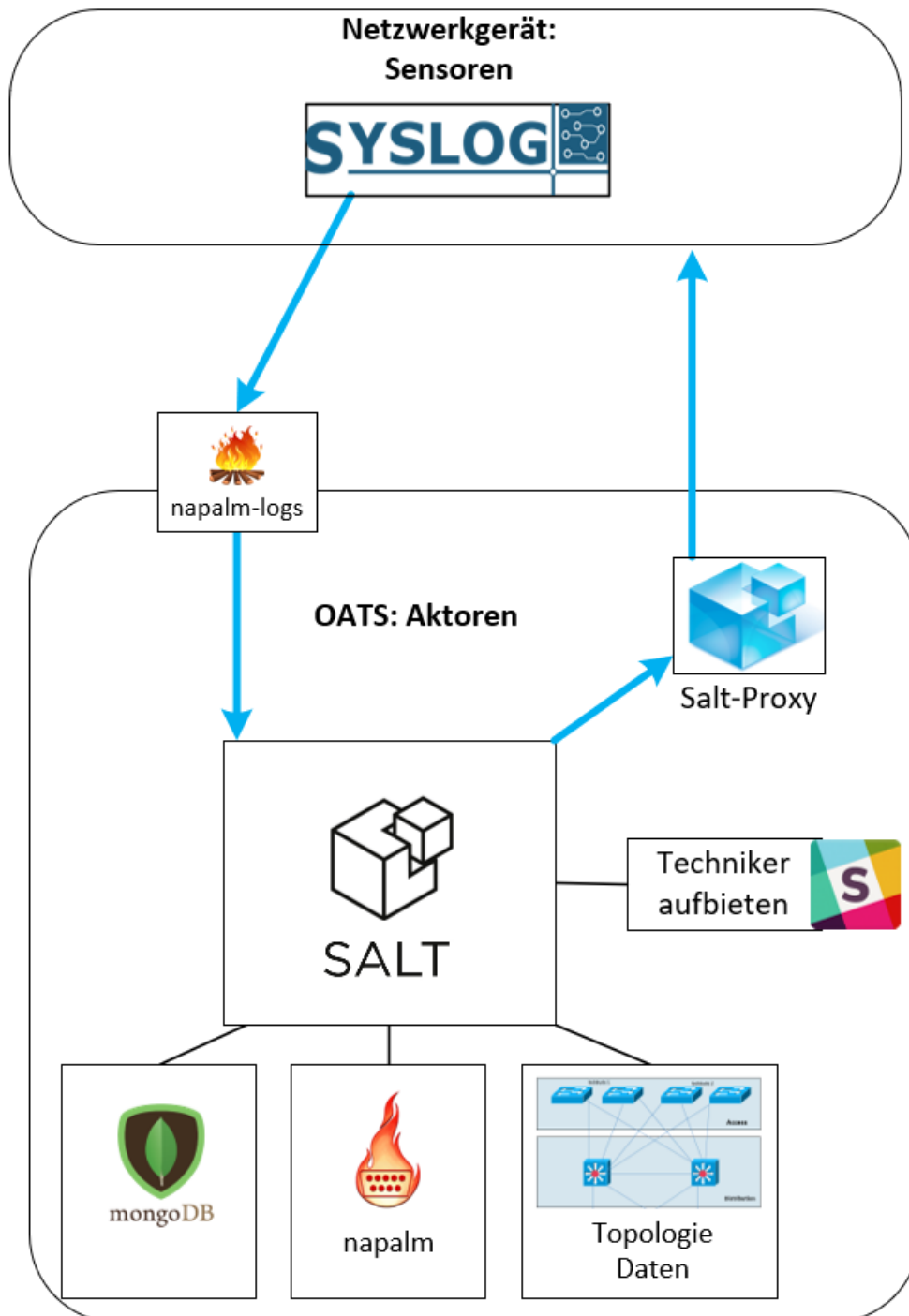


Abbildung 50: Architektur: Studienarbeit

In den folgenden Unterkapiteln werden die einzelnen Bereiche, welche zu Beginn der Studienarbeit erfasst wurden, aufgelistet und beschrieben, inwiefern diese durch die einzelnen gewählten Tools erfüllt werden können (siehe auch Doku zur Studienarbeit).



3.3.1 Erfassung / Sensoren

Napalm-logs wird nur einmal auf dem master *deployed* (ist aber nicht in Salt integriert) und leitet die Meldungen nicht direkt an den Salt-master weiter, sondern werden in einem Zwischenschritt vom Oatsclient verarbeitet. Der Oatsclient erstellt und versendet die Salt-Events.

3.3.2 Identifizierung

Die Identifizierung von Probleme geschieht an 2 Stellen.

Die erste Stelle ist der Oatsclient, der in der Lage ist, Meldungen von napalm-logs zu aggregieren, sodass schon früh erkannt werden kann, wo das Problem liegt.

Die zweite Stelle ist innerhalb von Salt. Dort wurde ein Python-file geschrieben, in welchem mittels Salt und Napalm Informationen gesammelt werden (z.B. per Pings oder Datenbankabfragen). Zu diesem Zweck ist die Netzwerktopologie in der eigenen MongoDB abgelegt.

3.3.3 Klassifizierung

Bei der Umsetzung wurde bemerkt, dass eine direkte Klassifizierung anhand der Severity-Levels keinen Sinn macht, da z.B. eine Interface-Down Meldung einen Level von 5 hat. Eine Interface-Down Meldung könnte aber bedeuten, dass nur das Interface runtergefahren wurde oder es könnte bedeuten, dass das Gerät nicht mehr online ist.

Hier müsste für die Zukunft ein neues Konzept entwickelt werden.

Die Klassifizierung in „vom System behandelbar“ und „Techniker benötigt“ geschieht fließend im gleichen Python-file wie die Identifizierung & Behebung. D.h. es wird nur eine Slack Meldung erstellt, falls das Problem nicht behoben werden konnte.

3.3.4 Behebung

Identifizierung & Behebung wird im gleichen Python-file durchgeführt (falls keine Aggregation stattfindet).

Dabei werden die Befehle nicht direkt vom Salt-master an die Geräte übermittelt, sondern werden vom Salt-master an den jeweiligen Salt-proxy (für jedes Netzwerkgerät existiert ein Salt-proxy) geschickt, welcher den Befehl an das Gerät weiterleitet.

Der Ablauf falls eine Aggregation durchgeführt wurde, ist der gleiche wie wenn keine stattgefunden hat. Nur ist es nicht nötig, vor der Behebung des Problems in Python noch weitere Informationen zu sammeln.

3.3.5 Überprüfung

Auch die Überprüfung geschieht direkt im gleichen Python-file. Dafür wird z.B. per Ping kontrolliert, ob ein Gerät welches vorher über ein bestimmtes Interface nicht erreichbar war, nun erreichbar ist.

3.3.6 Reporting/Logging

Das Reporting/Logging geschah per MongoDB. Allerdings wird nicht erst am Ende ein Log erstellt. Der Eintrag wird erstellt, sobald das Event im Oatsclient ausgelöst wird und wird bei jedem zusätzlich nötigem Schritt, um die neuen Informationen, ergänzt. So ist es möglich, falls eine Slack Meldung nötig ist, genaue Informationen zu den bisher getätigten Massnahmen zu liefern.



3.4 Architektur: Bachelorarbeit

Im Rahmen der Bachelorarbeit wurde das bestehende System wie folgt erweitert.

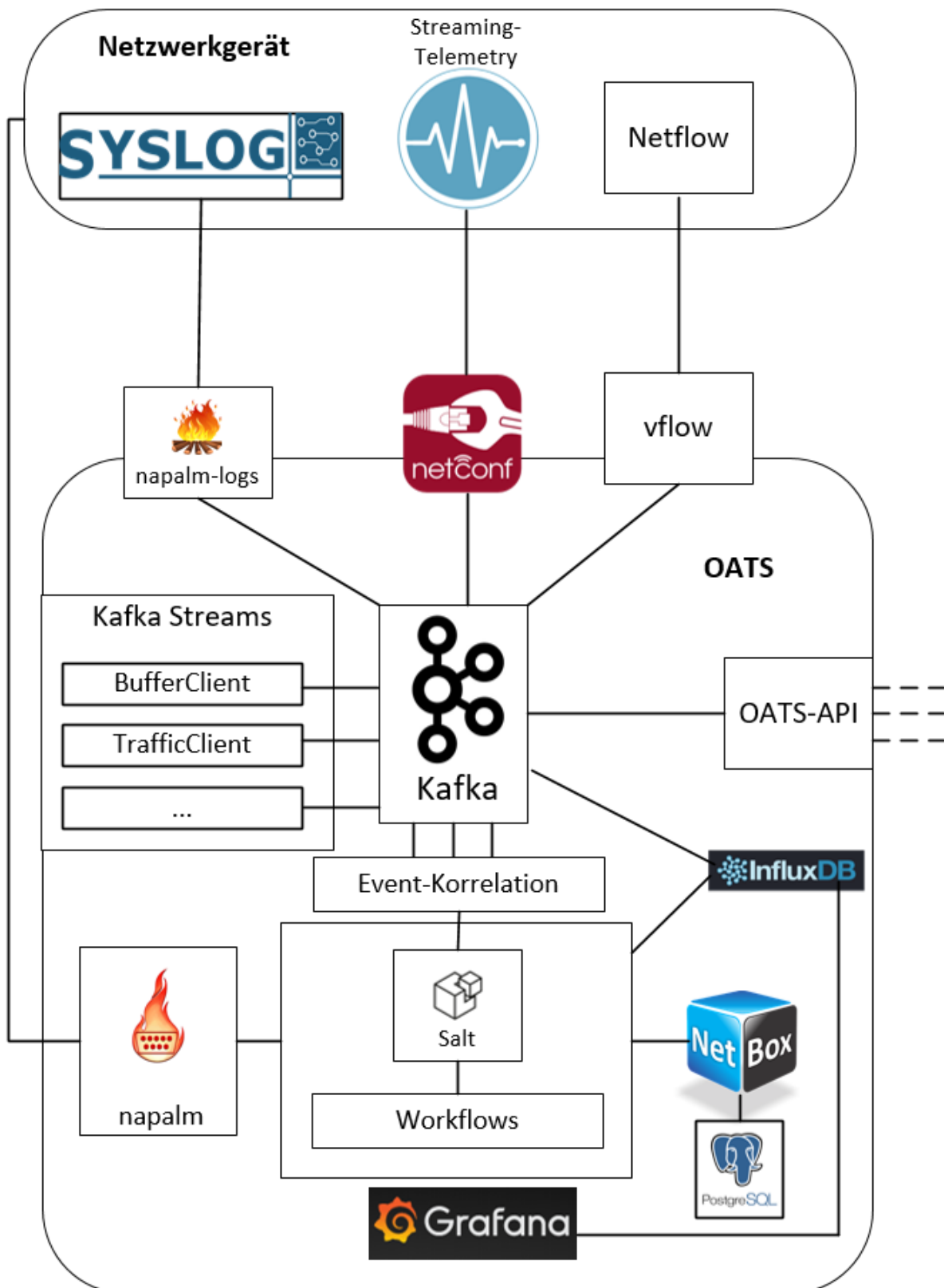


Abbildung 51: Architektur Bachelorarbeit



3.4.1 Netconf / Streaming Telemetry

Da in der Netzwerkumgebung Cisco IOS-XE Geräte zum Einsatz kommen, ist die eingesetzte Streaming-Telemetry Technologie NETCONF basiert. Dabei wird eine NETCONF-Session vom OATS-Server zu den jeweiligen Netzwerkgeräten eröffnet und aufrechterhalten. Über diese Session kann der Server gewisse YANG-Modelle abonnieren, für die er sich interessiert. Diese Daten werden in regelmässigen Abständen an den Server geschickt, wo ein Client die Daten entgegennimmt und sie in die Kafka-Queue mit einem entsprechenden Topic einreicht.

3.4.2 Netflow / vflow

Netflow wird direkt auf den Netzwerkgeräten konfiguriert und sendet regelmässig Flows an den OATS Server. Die Flows beinhalten Information zu der Grösse einer Datenübertragung, die Source- und Destinations-IP-Adressen, sowie die Source- und Destination-Port-Nummern der Übertragung. Auch diese Daten werden mittels vflow an Kafka-Topics übergeben. Vflow dient als zustandsloser Netflow-Collector, der die Daten direkt weiterleitet.

3.4.3 Syslog / Napalm-logs

Im Verlauf der Bachelorarbeit hat sich im Vergleich zum Stand der SA nur verändert, dass die Syslog-Messages von napalm-logs als Zwischenschritt in Kafka-Topics geschrieben werden, damit der Umgang mit den Daten innerhalb von OATS einheitlich ist.

3.4.4 OATS-API

Die OATS-API ist eine kleine Flask-App, welche eine REST-Schnittstelle für das Erstellen von Events, für OATS, anbietet. Dabei wurde darauf geachtet, dass das Datenformat für alle Quellen konsistent ist (mehr dazu im Kapitel 3.9, Datenspeicherung).

Über die OATS-API wäre es auch möglich, ein erweitertes UI für die Erstellung/Bearbeitung von Events zu entwickeln. Diese könnte man direkt in eine eventuell existierende Webseite oder WebApp einbinden. Als Hilfe und Dokumentation haben wir mit der Flask Erweiterung Flasgger eine API Dokumentation erstellt. Auf die Verwendung dieser gehen wir in der Benutzungsanleitung weiter ein.

3.4.5 Kafka

Kafka übernimmt in OATS die Rolle der Message-Queue und eines temporären Datenspeichers. Dabei existieren zwei unterschiedliche Arten von Kafka Topics:

- **Daten-Topics:** Topics, in welchen Daten für die weitere Verarbeitung (z.B. in Kafka-Streams) zwischengespeichert werden.
- **Event-Topics:** Topics, in welchen Events eingereicht werden. Diese Events benötigen keine weitere Datenaufbereitung und werden noch evtl. mit anderen Events korreliert.

Die Streaming-Telemetry Daten werden demnach in Daten-Topics eingeschleust. Für jedes Streaming-Telemetry Daten-Topic existiert ein Kafka Streams Java-Client, welcher die Daten verarbeitet und analysiert. Nach der Evaluation entscheidet jeder Client, ob ein Event eingetreten ist und wenn eines eingetreten ist, schreibt er dieses in ein Event-Topic.

Da jede syslog-message für sich schon ein Event ist, müssen diese nicht noch zusätzlich durch einen Kafka Streams Client evaluiert werden, sondern werden direkt in Kafka Event Topics geschrieben.

3.4.6 Kafka-Streams

Kafka-Streams ist ein Zwischenschritt bei dem die Streaming-Telemetry Daten analysiert werden. Dabei wird ein Wert kontinuierlich mit dem vorherigen Wert verglichen. Weicht der neueste Wert signifikant von dem vorherigen Wert ab, werden die Daten von diesem Wert als Event an ein Event-Topic weitergeleitet.



3.4.7 Event-Korrelation

Die Event-Korrelation geschieht in Kafka-Consumers, welche Records aus einem Event-Topic abarbeiten und eine der entwickelten Korrelations-Funktionen benutzen. Mit diesen Funktionen ist es möglich, Events zu aggregieren oder zu komprimieren. Wird nicht korreliert, so werden die Events direkt an Salt übertragen und es wird ein Troubleshooting-Workflow gestartet. Wird aggregiert oder komprimiert, wird auch mindestens ein Event an Salt geschickt, aber es ist möglich, dass statt 4 Events nur noch 1 Event übermittelt wird.

Innerhalb der Workflows werden z.T. auch noch Daten korreliert. So werden in einem Workflow der auf Streaming-Telemetry Daten basiert, zusätzlich die Netflow-Daten der gleichen Zeit analysiert, um auf den Ursprung des Events schliessen zu können.

3.4.8 InfluxDB

Event Daten, welche über Kafka von den Netzwerk-Sensoren geliefert werden, speichern wir in der Timeseries-Datenbank InfluxDB ab. Die Event Daten haben einen allgemeinen Teil, der bei allen Sensoren gleich ist, und einen Data Teil der jeweils unterschiedliche, sensorspezifische Informationen enthält.

Die Daten werden von einem Workflow zur Korrelation von Streaming-Telemetry- und Netflow-Daten verwendet. Zusätzlich werden zur Visualisierung kontinuierlich von Grafana ausgelesen.

3.4.9 Netbox

Die Netzwerkdaten werden über Netbox erfasst und in einem Web-UI angezeigt. Dort ist ersichtlich welche Geräte mit welchen verbunden sind und wie das gesamte Netzwerk zusammenhängt. Zudem erlaubt es, bei der Eingabe oder Änderungen, mögliche Fehler ersichtlich zu machen. Daten welche wir in einem Workflow aus der Topologie benötigen, können einfach über eine REST API abgefragt werden. Im Hintergrund verwendet Netbox eine PostgreSQL Datenbank, wo wir zudem Daten über die Problemfälle speichern. Sowohl Netbox, als auch die Postgres-Datenbank, sprechen wir über eigene OATS Python Packages an.

3.4.10 Grafana

Um ein attraktives und relevantes Dashboard zu haben, werden Daten aus der InfluxDB, von Grafana abgefragt, welche diese zusammenfasst und in Metriken darstellt.

Dies erlaubt einen schnellen Überblick über das System und ein einfaches Monitoring der Services.

3.4.11 Napalm & Salt

Napalm und Salt übernehmen immer noch die gleiche Funktion, innerhalb von OATS, wie während der Studienarbeit.

Salt reagiert auf Events mit dem richtigen Workflow-Skript und Napalm erlaubt es uns Informationen im Netz zu sammeln und Konfigurationen auf die Geräte zu laden.



3.5 Abläufe

Nachfolgend werden hier interne Abläufe in OATS niedergeschrieben. Dabei werden die einzelnen Schritte der Übersicht halber z.T. leicht verallgemeinert.

3.5.1 Syslog basierter Workflow

Ein beispielhafter Ablauf eines Troubleshooting-Workflows, der durch eine Syslog-Message ausgelöst wurde, sieht folgendermassen aus:

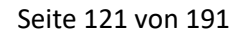
1. Ein Interface fährt runter, eine syslog-message mit allen Daten (in plaintext) wird an den OATS-Server gesendet.
2. Napalm-logs, welches auf dem Server läuft, nimmt diese message entgegen, extrahiert die Daten und schreibt diese Daten in ein Kafka Topic (das Kafka Topic wird über den Namen des Fehlers der syslog-message identifiziert).
3. Das oben genannte Topic wird von einem Kafka-Consumer kontinuierlich ausgelesen und erkennt, dass sich ein neues Event im Topic befindet. Der Consumer startet eine Aggregation unterschiedlicher Events, um genauer erfahren zu können, wo der Ursprung des Events liegt.
 - a. Es sind im Zeitraum der Aggregation keine zusätzlichen Events eingetroffen, d.h. der Ursprung liegt in einem einfachen Interface-Down Event.
 - b. Es sind im Zeitraum der Aggregation mehrere Interface-Down und Interface-Up Events eingetroffen, d.h. der Ursprung des Problems kann auf ein Port-Flapping reduziert werden.
4. Nach der Aggregation wird ein entsprechendes Event an Salt gesendet, welches alle relevanten Daten beinhaltet und die Daten werden in die InfluxDB geschrieben.
5. Je nach dem welches Event (Port-Flapping oder Interface-Down) an Salt gesendet wurde, wird ein anderes Troubleshooting durchgeführt, dass das Problem löst.
 - a. Das Problem lag bei einem einfachen Interface down Event, d.h. das betroffene Interface wird lediglich einmal runtergefahren und wieder hochgefahren.
 - b. Das Problem ist ein Port-Flapping auf dem betroffenen Interface, d.h. das betroffene Interface wird für einen gewissen Zeitraum (z.B. 20sec) runtergefahren und danach wieder hochgefahren.
6. Nach den Massnahmen wird mittels Pings-Befehlen verifiziert ob das betroffene Interface wieder erreichbar ist.
 - a. Das Interface ist erreichbar, der Troubleshooting-Workflow war erfolgreich und wird somit abgeschlossen.
 - b. Das Interface ist nicht erreichbar, der Troubleshooting-Workflow war nicht erfolgreich. Somit wird per Slack ein Techniker benachrichtigt, der das Problem manuell lösen muss.



3.5.2 Streaming-Telemetry basierter Workflow

Ein beispielhafter Ablauf eines Troubleshooting-Workflows der durch Streaming-Telemetry Daten ausgelöst wurde sieht folgendermassen aus:

1. OATS baut eine Netconf-Session zu allen Hosts des Netzwerks auf und hält diese aufrecht. Direkt danach teilt er den Geräten über diese Session mit, welche Daten er in regelmässigen Zeitabständen (z.B. 10 Sekunden) erhalten möchte. Die Daten, die OATS angefordert hat, basieren in diesem Fall auf dem «interfaces-state» Yang-Modell.
2. Alle 10 Sekunden werden von allen Netzwerkgeräten über die Netconf-Session Daten an den OATS-Server gesendet. Die Daten beinhalten die IP des Hosts, die Namen aller Interfaces des Hosts und den derzeitigen Zähler, der gedropten Pakete von jedem Interface.
3. OATS nimmt diese Daten entgegen und schreibt sie in ein Daten Topic mit dem Namen «interfaces-out-discards».
4. Der in OATS integrierte Kafka-Streams Client liest dieses Topic kontinuierlich aus. Dabei vergleicht er jeweils den neuesten Wert der gedropten Pakete eines Interfaces mit dem vorigen Wert desselben Interfaces.
5. Die Evaluation der beiden Werte ergibt folgendes:
 - a. Die beiden Werte sind gleich oder überschreiten den gesetzten Schwellenwert für ein Event nicht. Es passiert nichts.
 - b. Die Differenz der beiden Werte ist grösser als der gesetzte Schwellenwert. Es wird ein Event in das Event-Topic «out-discards-events» geschrieben.
6. Ein Kafka-Consumer, welcher dieses Topic ausliest, nimmt dieses Event entgegen und startet eine Komprimierung der Events der gleichen Art für 10 Sekunden. Dies wird gemacht, damit Salt nicht mit Events überflutet wird.
7. Sind die 10 Sekunden verstrichen, so wird ein Event dieser Art an Salt weitergeleitet.
8. Aufgrund dieses Events startet Salt den Workflow zum Lösen dieses Problems.
9. Zu Beginn des Workflows werden die aktuellsten Netflow-Daten analysiert, um den Flow ausfindig zu machen, der für die gedropten Pakete verantwortlich ist. Der richtige Flow ist derjenige, der vom gleichen Host stammt wie das ursprüngliche Event und bei dem die Anzahl der Pakete hoch ist (>15'000 in diesem Fall).
10. Aus dem Flow werden die Source- und Destination-IP, sowie die Destination-Port-Nummer entnommen. Diese Informationen werden dann benutzt um eine ACL auf das Gerät zu laden, welche den Verkehr von Source- zu Destination-IP mit diesem Destination-Port auf 8000 Kbit/s (CIR = 8000) reduziert.
11. Nach einer gewissen Zeitdauer (z.B. 5min) wird diese ACL wieder vom Gerät entfernt.



```

classDiagram
    class napalm_logs {
        class oats_napalm_logs_client
    }
    class oats_api {
        class app {
            +create_event()
        }
    }
    class telemetry {
        class netconf_telemetry {
            +process_host_config()
            +__create_subscriptions()
        }
        class oats_telemetry {
            +main()
        }
        class kafka_streams {
            +start_kafka_streams()
        }
        class model {
            class OATSConfig {
                +get_host_configs() list(Host)
                +get_telemetry_subscriptions() list(TelemetrySubscription)
            }
            class TelemetrySubscription
            class Host
        }
    }
    class streaming_clients {
        <<java>>
        class OATSStatisticsClient {
            +main()
        }
        class processor {
            class StatisticTransformer {
                +transform()
            }
        }
        class serde {
            class JSONPOJOSerializer {
                +serialize()
            }
            class JSONPOJODeserializer {
                +deserialize()
            }
            class StatisticsDeserializer {
                +deserialize()
            }
        }
        class model {
            class OATSArgs
            class Operator
            class Statistics
            class Statistic
        }
    }
    class kafka {
        class generic_kafka_consumer {
            +consume_kafka() void
        }
        class oats_api_consumer
        class oats_event_consumers
        class oats_netflow_consumer
        class helpers {
            class oats_correlate {
                +aggregate_identical()
                +aggregate_distinct()
                +compress()
            }
            class EventProcessor {
                +process_event()
            }
            class utils {
                +get_n_of_events_and_root_host()
                +extract_record_data()
            }
        }
    }
    class salt {
        class runners {
            class tshoot {
                +ifdown()
                +ospf_nbr_Down()
                +out_discards_exceeded()
                +port_flap()
            }
            class oattsalthelpers {
                +post_slack()
                +ping()
                +if_noshutdown()
                +if_shutdown()
                +ospf_shutdown()
                +ospf_noshutdown()
                +check_device_connectivity()
                +count_event()
                +apply_policy()
                +remove_policy()
                +wait_for_event()
                +get_src_flow()
            }
        }
        class database_access {
            class oatsmysql {
                +create_case()
                +update_case()
                +take_case()
                +close_case()
                +delete_case()
                +get_solutions_as_string()
            }
            class oatsnetbox {
                +get_ospf_neighbors()
                +get_vrf_ip()
                +get_hostname()
                +get_interface_neighbor()
                +get_neighbors()
            }
            class oatsinflux {
                +write_event()
                +get_type_data()
            }
        }
    }
    napalm_logs --> kafka
    oats_api --> kafka
    oats_telemetry ..> netconf_telemetry
    oats_telemetry ..> kafka_streams
    kafka_streams ..> model
    streaming_clients ..> kafka
    kafka ..> salt
    tshoot ..> oattsalthelpers
    oattsalthelpers ..> database_access
  
```

Innerhalb der Packages wurden nur die Abhängigkeiten (Pfeile) eingetragen, wo es explizit Sinn macht. Ansonsten kann davon ausgegangen werden, dass die Abhängigkeiten von oben nach unten verlaufen.



3.6.1 napalm-logs

Enthält ein Skript, welches dauerhaft auf dem OATS-Server läuft und auf Nachrichten von napalm-logs wartet. Es ist als Systemd-Daemon aufgesetzt (für mehr Informationen zu Systemd siehe Kapitel 3.10, Prozesse und Threads).

Klasse	Beschreibung
oats_napalm_logs_client	Nimmt die Syslog-Messages, mitsamt den Daten aus napalm-logs entgegen und schreibt sie entsprechend ihrem Event-Namen (z.B. INTERFACE_CHANGED) in ein Kafka-Topic.

3.6.2 oats_api

Enthält eine Flask-App, welche dauerhaft auf dem OATS-Server auf Nachrichten, welche über die REST-Schnittstelle übermittelt werden, wartet und als Systemd-Daemon aufgesetzt ist.

Klasse	Beschreibung
app	Bietet eine REST-Schnittstelle, für das Kreieren von Events in OATS an. Die Daten müssen in der gleichen Form übergeben werden, wie sie auch bei den anderen Sensoren verwendet wird. Das Datenformat wird im Kapitel 3.9 (Datenspeicherung) genauer beschrieben.

3.6.3 telemetry

Enthält den Streaming-Telemetry Hauptprozess und ist dafür verantwortlich, dass alle Streaming-Telemetry relevanten Prozesse gestartet werden.

Klasse	Beschreibung
oats_telemetry	Liest die Konfigurationsdatei, die als <i>/etc/oats/config.yaml</i> abgelegt ist, aus und startet basierend auf dieser Konfiguration eine gewisse Anzahl anderer Prozesse bzw. Threads.
netconf_telemetry	Ist dafür verantwortlich die Netconf-Sessions zu den jeweiligen Hosts aufzubauen und startet gleichzeitig für jedes YANG-Modell, welches abonniert wurde, einen zusätzlichen Thread pro Host. Nimmt anschliessend die Daten aus der Netconf-Session entgegen und schreibt sie in ein Kafka-Topic, welches auch über die Konfigurationsdatei angegeben wird.
kafka_streams	Startet einen Kafka-Streams Client und übergibt dabei die Parameter, welche aus der Konfigurationsdatei stammen.
OATSConfig	Eine Klasse welche von <i>oats_telemetry</i> benötigt wird, um die Konfigurationsdatei auszulesen.
Host	Reine Datenhaltungsklasse, welche alle Informationen eines Hosts beinhaltet. Wird von <i>OATSConfig</i> verwendet, um die Host-Konfigurationen zu speichern.
TelemetrySubscription	Auch eine Datenhaltungsklasse, welche alle Informationen zu den Subscriptions beinhaltet. Wird von <i>OATSConfig</i> verwendet um die Subscription-Details zu speichern.



3.6.4 streaming_clients

Enthält das Java-Projekt, in welchem der OATS Kafka-Streams Client entwickelt wurde.

Klasse	Beschreibung
OATSStatisticsClient	Der Kafka-Streams Main-Thread, welcher aus dem, in der Konfigurationsdatei, angegebenen Input-Topic liest und die Daten in den Records analysiert (stateful, erzeugt, sobald er ein Event entdeckt, ein Event-Record und schreibt dieses in ein Output-Topic, auch in der Konfiguration angegeben.)
StatisticTransformer	Hält einen «in-memory»-Speicher, in welchem die alten Werte mit den neuen Werten verglichen werden können. Wird vom Main-Thread benötigt um Events zu erkennen.
JSONPOJODeserializer	Deserializer, um die JSON-Objekte aus den Kafka-Topics deserialisieren zu können.
JSONPOJOSerializer	Serializer, um die internen Java-Objekte wieder in JSON serialisieren zu können.
StatisticsDeserializer	Deserializer, um die String-Objekte, welche vom JSONPOJODeserializer geliefert werden, in ein «Statistics» Java-Objekt deserialisieren zu können.
OATSArgs	Datenhaltungsklasse, welche die Argumente, welche dem Main-Thread beim Programmstart übergeben werden, enthält.
Operator	Enum, welcher die Operator-Strings in <i>OATSArgs</i> auf die richtigen, bei der Evaluation verwendeten Operatoren, mappt. So wird aus «greater_than» der Operator > (grösser als).
Statistics	Datenhaltungsklasse, welche den Event-Timestamp und eine Liste von <i>Statistic</i> -Objekten enthält.
Statistic	Datenhaltungsklasse, welche eine einzelne Statistik enthält. Wird benötigt, damit die einzelnen Statistiken verglichen werden können.



3.6.5 kafka

Enthält diverse Kafka-Consumer und Helfer-Methoden für diese Consumer.

Klasse	Beschreibung
generic_kafka_consumer	Ein generischer Kafka-Consumer, der eine Reihe von Übergabeparametern entgegennimmt. Aus diesen Übergabeparametern wird entschieden, aus welchem Kafka-Topic gelesen werden soll und ob die Records aus diesem Topic korreliert werden sollen oder nicht. Wird nicht korreliert, so werden die Daten aus den Records an Salt weitergeleitet und so ein Troubleshooting-Workflow gestartet.
oats-api_consumer	Kafka-Consumer, welcher die Records aus dem «oats-api»-Topic ausliest und in die InfluxDB schreibt. Müsste leicht angepasst werden, wenn auch ein Troubleshooting gestartet werden soll.
oats-netflow_consumer	Kafka-Consumer, welcher die Records aus dem «oats-netflow» Topic ausliest (wird vom Service «vflow» gefüllt) und in die InfluxDB schreibt.
oats-event_consumers	Steht stellvertretend für mehrere weitere Kafka-Consumer, welche aus verschiedenen Syslog- und Streaming-Telemetry Kafka-Topics lesen (Diagramm wäre sonst zu unübersichtlich geworden). Die Events aus diesen Topics werden mehrheitlich noch korreliert.
EventProcessor	Alle Events welche ein Troubleshooting starten und/oder in die InfluxDB geschrieben werden, werden mithilfe dieser Klasse verarbeitet. Der wichtigste Punkt ist dabei, dass damit die Form der Events innerhalb von OATS einheitlich ist, unabhängig von der Quelle der Events. Wird innerhalb der Korrelationsfunktionen und der Kafka-Consumer benutzt, welche keine Korrelation durchführen.
oats_correlate	Mithilfe der 3 Methoden innerhalb dieser Klasse, können Events aggregiert oder komprimiert werden, sie wird von den meisten Kafka-Consumer benutzt. Am Ende jeder Korrelation wird mindestens ein Event an Salt gesendet (mittels EventProcessor). Zusätzlich wird, zu Beginn der Korrelation, noch jedes Event in die InfluxDB geschrieben, damit keine Daten in der Datenbank fehlen.
utils	Enthält zwei nützliche Utility-Methoden, welche an mehreren Stellen verwendet werden.

3.6.6 salt

Enthält die Troubleshooting-Workflows, welche durch Salt ausgeführt werden und einige Helfermethoden, welche in den Workflows verwendet werden.

Klasse	Beschreibung
tshoot	Die tshoot Klasse beinhaltet vier Funktionen, welche alle einen unterschiedlichen Problemfall behandeln (OSPF_NEIGHBOR_DOWN, INTERFACE_DOWN, PORT_FLAPPING, OUT_DISCARDS_EXCEEDED). Diese Funktionen können entweder über den Salt-reactor (über den richtigen Event-Namen) oder direkt über die Salt-cli aufgerufen werden. Wird im Zusammenhang mit OATS ausschliesslich über den Salt-reactor ausgeführt.
oatssalthelpers	Diese Klasse beinhaltet diverse Helfer-Funktionen, um von Salt und den Workflows auf das Netzwerk zuzugreifen oder Salt-interne Funktionen benutzen zu können.



3.6.7 oatsnetbox

Regelt die Zugriffe auf die Daten von Netbox. Diese Daten beinhalten hauptsächlich Informationen zu den Netzwerkgeräten.

Klasse	Beschreibung
oatsnb	Die Klasse oatsnetbox ist in der Lage eine Verbindung zu Netbox aufzubauen, die relevanten Daten zu lesen und die Verbindung wieder zu schliessen. Für die Verwendung in OATS können folgende Daten ausgelesen werden. • Die Management IP, falls man nur den Hostnamen eines Geräts kennt. • Den Hostnamen eines Geräts, falls man nur die Management IP kennt. • Alle Nachbarn eines Geräts. • Ein spezifischer Nachbar, bei dem nur das Interface bekannt ist, mit welchem ein Host zu diesem Nachbar verbunden ist. • Alle Nachbarn eines Hosts, die sowohl OSPF verwenden und auch in der gleichen OSPF Area sind.

3.6.8 oatssql

Enthält alle Funktionen mit der auf die PostgreSQL Datenbank zugegriffen werden, um dort Daten abzurufen. Bei diesen Daten handelt es sich um Informationen zu den Problemfällen.

Klasse	Beschreibung
oatssql	Die Klasse oatssql beinhaltet Funktionen, mit welchen Verbindungen zur PostgreSQL Datenbank auf und abgebaut werden können. Zusätzlich sind hier alle Funktionen bei denen es um Problemfälle geht enthalten. So kann ein Fall erstellt, überarbeitet, einem Techniker zugewiesen, abgeschlossen und gelöscht werden.

3.6.9 oatsinflux

Hier sind alle Funktionen enthalten bei denen es um die InfluxDB geht. Es wird jeweils mit Events gearbeitet, also Ereignisse innerhalb des Netzwerks, welche von einem Sensor entdeckt wurden. Dies sind Timeseries-Daten, haben also alle jeweils einen Timestamp.

Klasse	Beschreibung
oatsinflux	In der Klasse findet man alle Funktionen, welche mit der InfluxDB interagieren. Als erstes regelt eine Funktion den Verbindungsaufbau zur Datenbank. Ein Event kann mittels der write_event Funktion in die Datenbank geschrieben werden. Diese sortiert die Events nach Typen, je nach Typ werden andere Daten ausgelesen und die Daten entsprechen formatiert. Wie diese Events genau aussehen, ist im Kapitel 3.9 genauer beschrieben. Es wird zwischen vier Event Typen unterscheiden: Syslog, Netflow, Streaming-Telemetry und API. Schliesslich kann diese Klasse, mit der get_type_data funktion, auch Daten aus der Datenbank auslesen. Diese liefert alle Daten, eines Event Typs, in einem angegebenen Zeitfenster.



3.7 Schnittstellen

3.7.1 OATS <> User

Ein User von OATS kann über 5 Schnittstellen mit dem System interagieren:

- **OATS-API:** Kann benutzt werden um manuell Events in das System einzuschleusen und neue (externe) Sensoren an das System anzuschliessen. Bietet dafür eine REST-Schnittstelle an.
- **OATS-Konfigurationsdatei:** Der User kann direkt in YAML beschreiben, welche Daten von welchen Hosts per Streaming-Telemetry geliefert werden sollen und was mit diesen Daten gemacht werden soll.
- **Netbox:** Bietet ein UI an, über welches die Netzwerktopologie in eine PSQL-Datenbank geschrieben werden kann, welche von OATS beim Troubleshooting als „Source of Truth“ benötigt wird.
- **Salt:** Salt bietet eine Abstraktion per CLI für gewisse Salt-Funktionen, z.B. für das Handling der Salt-master, -minions und -proxies.
- **Grafana:** Über Grafana können nützliche Statistiken, aus der InfluxDB generiert und dargestellt werden. Der User bekommt so einen schnellen Überblick über den Zustand des Netzes.

3.7.2 OATS <> Sensoren

Streaming-Telemetry

Für das Senden der Streaming-Telemetry Daten wird von OATS eine Netconf-Session zu dem jeweiligen Netzwerkgerät aufgebaut. Die Netconf-Session basiert auf TCP und läuft in OATS über Port 830.

Netflow

Cisco Netflow sendet die Netflow-Daten per UDP an Port 4729 auf dem OATS-Server, wo der Service vflow auf diesen horcht und die Daten anschliessend in ein Kafka-Topic schreibt.

napalm-logs

Die Netzwerkgeräte haben Syslog konfiguriert, diese werden auf einen definierten Port (33333) per UDP an den Server auf dem napalm-logs läuft, gesendet.

3.7.3 OATS <> Netzwerkgeräte

Um Daten zu sammeln und Befehle direkt auf den Geräten auszuführen, greift OATS (über Salt) mittels napalm auf die Geräte zu. Napalm benutzt für die Zugriffe verschiedene Protokolle/Tools. Für Cisco IOS, welches in der Testumgebung der SA eingesetzt wurde ist dies SSH.

Sollten in Zukunft Geräte mit anderen Betriebssystemen als Cisco IOS angesteuert werden, würde sich diese Schnittstelle ändern. Dies ist aber für OATS nicht relevant, da dies von napalm erledigt wird, welches inhärent herstellerunabhängig ist.



3.7.4 Interne Schnittstellen

Kafka

Der mit Abstand grösste Teil der internen Kommunikation, zwischen den verschiedenen Prozessen, geschieht über Kafka-Topics. Der grösste Vorteil davon ist, dass damit jede Komponente theoretisch auf unterschiedliche Server verteilt werden kann. Angepasst werden müsste dafür lediglich die Kafka-Konfiguration (im Moment läuft alles über localhost).

ZeroMQ

Über ZeroMQ werden die Salt-Events an den Salt-master geschickt und die Kommunikation zwischen napalm-logs und dem oats_napalm_logs_client läuft auch über ZeroMQ.



3.8 Deployment

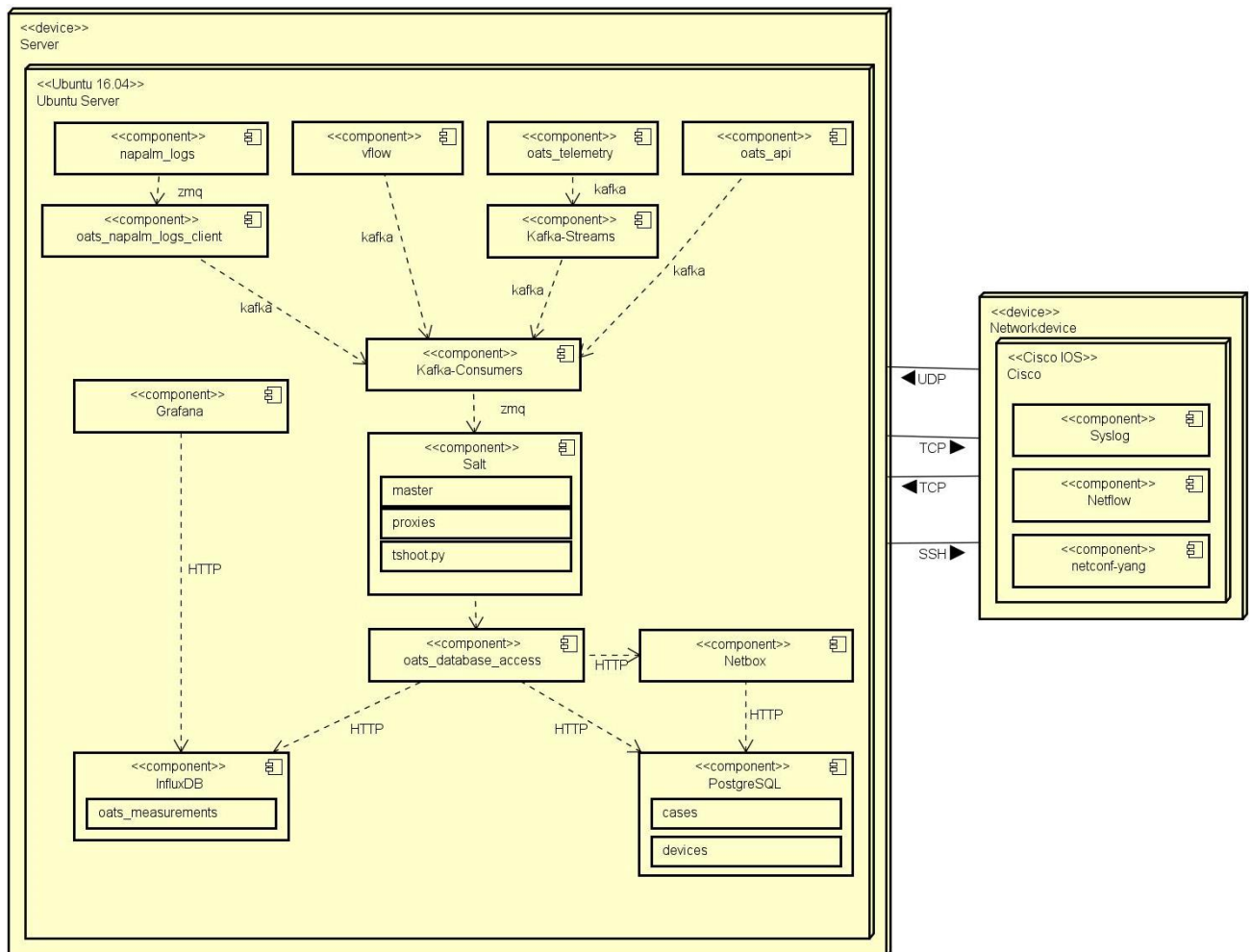


Abbildung 53: Deployment Diagramm

3.8.1 Server

In obiger Abbildung ist ersichtlich, wie OATS während der Bachelorarbeit aufgesetzt wurde. Da es die Entwicklung massiv vereinfacht hat, wurden alle Komponenten auf dem gleichen Server aufgesetzt. Es wäre aber denkbar, und bei grossen Netzwerken zu empfehlen, OATS auf mehrere Server zu verteilen.

Da sämtlicher interner Verkehr entweder über Kafka, ZeroMQ, oder Http verläuft, wäre es theoretisch denkbar, jede Komponente einzeln auf einem Server aufzusetzen. OATS kann somit sehr gut horizontal hochskaliert werden.

Die einzelnen Prozesse werden im Kapitel 3.10 (Prozesse und Threads) vorgestellt.

3.8.2 Netzwerkdevice

Bei den Netzwerkgeräten handelt es sich um Cisco Router und Switches. Auf diesen muss Syslog aktiviert sein um Meldungen übertragen zu können. Zudem laufen Netflow und Streaming Telemetry auf diesen Geräten, um Daten über das Netzwerk und den Traffic zu erhalten.



3.9 Datenspeicherung

3.9.1 Device

Informationen zu den einzelnen Netzwerkgeräten wurde während der Studienarbeit in einer MongoDB mit eigenem Datenmodell gespeichert. Da für die Bachelorarbeit die Datenspeicherung überarbeitet wurde, wird nun Netbox verwendet, welche im Hintergrund Daten in einer PostgreSQL Datenbank speichert.

Für die umgesetzten Workflows werden auf die unten aufgelisteten Informationen in Netbox zugegriffen. Man kann in Netbox problemlos weitere Netzwerk-, Protokoll- und Geräte-Informationen abspeichern. Da diese jedoch für die umgesetzten Workflows nicht benötigt wurden, sind diese hier nicht erwähnt.

Sites: Falls man mehrere Netzwerke an verschiedenen Standorten hat, kann man die einzelnen Geräte in verschiedenen Sites speichern.

Role: Die Devices haben verschiedene Rollen wie etwa Access Switch oder Core Router, um deren Aufgabe im Netzwerk zu erfassen.

Hostname: Als erstes wird für alle Devices der Hostname gespeichert, diese sind dann jeweils einer Site zugeordnet. Dieser Hostname entspricht dem in Salt verwendeten Hostnamen, für die einzelnen Geräte und deren Salt-Minions. Der Hostname wird in den Workflows hauptsächlich als Filter verwendet, um Informationen zu einem spezifischen Gerät zu erhalten.

Management IP: Der OATS Server ist zu den einzelnen Devices über ein Management-Netzwerk verbunden. Dies ist ein, vom regulären Netzwerk, welches die einzelnen Devices untereinander verbindet, getrenntes eigenes Netzwerk.

Interfaces: Die Interfaces der Geräte werden ebenfalls abgespeichert. Diese enthalten folgende Informationen:

Name: Der Plaintext Name des Interfaces, wie etwa „GigabitEthernet1“. Der Name des Interfaces wird, in unseren Workflows, zusammen mit dem Hostnamen des Geräts verwendet, um die Nachbarn eines Geräts zu erhalten.

IP Adresse: Die IP-Adresse des Interfaces.

Connection: Bei den Interfaces welche mit einem anderen Gerät verbunden sind, wird gespeichert, welches Gerät der Nachbar ist und über welche Interfaces sie verbunden sind. Für einen Workflow muss man in der Lage sein zu ermitteln, welche Nachbarn ein Gerät hat.

OSPF AREA: Bei einem Workflow kann man Probleme beheben, welche sich auf das OSPF Protokoll beziehen. Deshalb man ermitteln können, bei welchen Verbindungen, zwischen welchen Geräten, OSPF als Routing Protokoll verwendet wird. Zusätzlich muss man noch erfassen, um welche OSPF Area es sich handelt, da zwei Geräte welche miteinander verbunden sind zwar beide OSPF verwenden können, es aber sein kann, dass diese nicht in der selben Area liegen. Dies kann der Fall sein, wenn es sich beim Router um einen Area Border Router in Bezug auf OSPF handelt.

VLAN Nummer, Name und Modus: Informationen zu Vlan werden ebenfalls erfasst. Die Informationen zu den VLANs enthalten die Nummer und den Namen sowie der Modus, also etwa statisch oder dynamisch. Danach werden die VLANs einem Port zugewiesen.

Für neue Workflows kann es sein, dass neue Informationen erfasst und abgespeichert werden müssen. Zudem müssen der Zugriff und das Auslesen dieser Informationen geregelt werden.



3.9.2 Events

In der InfluxDB werden die Event Daten gespeichert. Die Sensoren, mit welchen Informationen aus dem Netzwerk gesammelt werden, kommen alle in einem unterschiedlichen Format und enthalten unterschiedliche Informationen. Zur sauberen Verarbeitung in OATS und Korrelierung dieser Daten, müssen sie in ein einheitliches Format gebracht werden. Dieses Format wurde eigens für OATS entwickelt, es besteht aus allgemeinen Feldern, welche bei allen Sensoren identisch sind. Sowie einem Data Feld, welches die von Sensor zu Sensor unterschiedlichen Daten enthält.

Die allgemeinen Felder, der Events, sind die folgenden:

Sensor_type: Beschreibt welcher Netzwerk Sensor diese Daten geliefert hat. Es werden Syslog, Netflow, Streaming-Telemetry und die API unterschieden. Dieses Feld definiert zudem, innerhalb von InfluxDB, die Measurement-Tabelle, in welcher die Events gespeichert werden.

Event_name: Der Event Name beschreibt das erhaltene Event.

Timestamp: Die Timestamps geben Auskunft darüber wann ein Event vorgefallen ist. Das Format dieser Timestamps ist Epoch-UTC. Je nach Sensor Typ ist die Genauigkeit hier Sekunden oder Millisekunden. Innerhalb von Influx könnten diese bis auf die Nanosekunde genau gespeichert werden.

Host: Gibt Auskunft darüber, von welchem Host das Event gesendet wurde.

Severity: Die Severity sagt aus wie schwer dieser Fehler das System beeinträchtigt. Innerhalb dieser Bachelorarbeit wurde diese Funktionalität erst konzipiert, jedoch noch nicht umgesetzt.

Data: Im Data Feld werden Daten gespeichert die spezifisch sind auf die jeweiligen Sensoren. Sie unterscheiden sich stark voneinander und liefern unterschiedliche Informationen.

Syslog

Syslog wurde als Sensor bereits in der vorgegangenen Studienarbeit. In der Bachelorarbeit wurden die Sensoren erweitert. In Influx hat Syslog alle die oben genannten Felder, spezielle Syslog Spezifische Informationen sind hier vermerkt. Das Influx Measurement für Syslog wurde „oats_timeseries_syslog“ genannt.

Timestamp: Das Syslog Timestamp wird angegeben auf die Millisekunde genau.

Severity: Da eine Syslog Meldung bereits eine Severity enthält, wird diese direkt übernommen.

Data: Bei Syslog wurde bisher mit „Interface Changed“ und „OSPF Neighbour“ Meldungen gearbeitet, weshalb sich das Syslog Data Feld bis jetzt auf die Eigenheiten dieser Meldungen beschränkt.

Genauer heisst dies, für „State Changed“ Meldungen erhält man ein „oper status“ Feld innerhalb der Meldung, der Wert dieses Feldes ist entweder „up“ oder „down“. Dies wird so übernommen und in die Influx Datenbank geschrieben. Bei den OSPF Meldungen erhält man ähnliche Informationen, als „adjacency-state“ Informationen, auch dieses sind entweder „up“ oder „down“. Zusätzlich erhält man hier auch noch eine „adjacency-state-change-reason“ welche den Grund beschreibt, wieso es zu dieser Änderung kam. Dieser kann zum Beispiel „DEAD“ sein. Bei diesem Beispiel erhält man zudem ein „adjacency-state-change-reason-message“, dieses könnte in diesem Fall „Dead timer expired“ lauten.

Neben diesen Informationen wird in die InfluxDB noch ein weiteres Feld geschrieben: „if_state_id“ für Interface changed Meldungen und „ospf_state_id“ für OSPF Meldungen. In diesen Feldern werden entweder 0 oder 1 geschrieben, um anzuzeigen ob das betroffene Interface UP oder DOWN ist. Dieses binäre ID Feld wird danach von Grafana abgefragt um den Status des Netzwerks zu visualisieren.



Netflow

Das Influx Measurement für Netflow wurde „oats_timeseries_netflow“ genannt.

Data: Das erhaltene Netflow Informationspaket enthält zwei Arten von Informationen. Die Header Informationen, wo etwa eine Sequenz Nummer drinsteht oder die Epoch-Sekunden Angabe. Die Epoch-Angabe wird im Timestamp-Feld angegeben um die Zeit zu speichern, an welcher der Flow aufgenommen wurde. Der zweite Teil ist ein DataSet, welches die genauen Informationen zum Flow enthält, eine genaue Erklärung dazu in der Tabelle [3] unten.

```
name: oats_timeseries_netflow
time                 1    10 11    12    14 152    153    2 4 5    6    61 7    8    event_name    host    severity type
-----
1526456897000000000 46 8 179 1.1.1.10 0 1526456883000 1526456883000 1 6 192 0x10 0 27270 1.1.1.9 netflow/*data 10.20.1.22 0 netflow
1526456910000000000 59 0 27270 1.1.1.9 8 1526456890000 1526456890000 1 6 192 0x18 1 179 1.1.1.10 netflow/*data 10.20.1.21 0 netflow
1526456933000000000 59 8 179 1.1.1.10 0 1526456914000 1526456914000 1 6 192 0x18 0 27270 1.1.1.9 netflow/*data 10.20.1.22 0 netflow
1526456940000000000 46 0 27270 1.1.1.9 8 1526456921000 1526456921000 1 6 192 0x10 1 179 1.1.1.10 netflow/*data 10.20.1.21 0 netflow
1526456952000000000 46 8 179 1.1.1.10 0 1526456937000 1526456937000 1 6 192 0x10 0 27270 1.1.1.9 netflow/*data 10.20.1.22 0 netflow
1526456958000000000 59 0 27270 1.1.1.9 8 1526456943000 1526456943000 1 6 192 0x18 1 179 1.1.1.10 netflow/*data 10.20.1.21 0 netflow
1526456982000000000 46 0 27270 1.1.1.9 8 1526456966000 1526456966000 1 6 192 0x10 1 179 1.1.1.10 netflow/*data 10.20.1.22 0 netflow
1526456988000000000 46 0 27270 1.1.1.9 8 1526456973000 1526456973000 1 6 192 0x10 1 179 1.1.1.10 netflow/*data 10.20.1.21 0 netflow
1526457006000000000 46 8 179 1.1.1.10 0 1526456989000 1526456989000 1 6 192 0x10 0 27270 1.1.1.9 netflow/*data 10.20.1.22 0 netflow
1526457013000000000 59 0 27270 1.1.1.9 8 1526456995000 1526456995000 1 6 192 0x18 1 179 1.1.1.10 netflow/*data 10.20.1.21 0 netflow
```

Abbildung 54: Netflow-Daten in der InfluxDB

Netflow Dataset Informationen		
I (Value ID)	V (möglicher Beispiel Inhalt)	Erklärung
1	46	IN_BYTES Incoming counter with length N x 8 bits for number of bytes associated with an IP Flow.
10	8	INPUT_SNMP Input interface index; default for N is 2 but higher values could be used
11	179	L4_DST_PORT TCP/UDP destination port number i.e.: FTP, Telnet, or equivalent
12	1.1.1.10	IPV4_DST_ADDR IPv4 destination address
14	0	OUTPUT_SNMP Output interface index; default for N is 2 but higher values could be used
152	1526456883000	Timestamp: first
153	1526456883000	Timestamp: last
2	1	IN_PKTS Incoming counter with length N x 8 bits for the number of packets associated with an IP Flow
4	6	PROTOCOL IP protocol byte
5	192	SRC_TOS Type of Service byte setting when entering incoming interface
6	0x10	TCP_FLAGS Cumulative of all the TCP flags seen for this flow
61	0	DIRECTION Flow direction: 0 - ingress flow, 1 - egress flow
7	27270	L4_SRC_PORT TCP/UDP source port number i.e.: FTP, Telnet, or equivalent
8	1.1.1.9	IPV4_SRC_ADDR IPv4 source address



Streaming Telemetry

Das Influx-Measurement für die Streaming Telemetry wurde „oats_timeseries_streaming“ genannt.

Timestamp: Das von der Streaming Telemetry erhaltene Timestamp ist auf die Millisekunde genau.

Data: Im Daten Feld, eines Streaming Telemetry Events, wird der Namen des Werts und der dazugehörige Wert angegeben. Diese Daten, zusammen mit dem bereits im allgemeinen Teil gelieferten Hostnamen, gibt Einsicht darüber, wieviele Pakete auf einem Interface gedroppt wurden.

API

Die API ist dazu gedacht, dass man auch manuell Events erstellen kann oder alte Network Monitoring Systeme an OATS anschliessen kann. Hier wurden nur wenige zusätzliche Änderungen für Influx und das Event an sich vorgenommen. Das Influx Measurement für die API wird „oats_timeseries_api“ genannt.

Data: Ein Einfaches 0 oder 1 wird hier als Data in die InfluxDB geschrieben. Um auszusagen, ob von der API her alles reibungslos läuft oder ob jemand eine Problemmeldung eingetragen hat. Für eine Verwendung in einem produktiven Umfeld, müsste dies noch weiter ausgebaut werden.

3.9.3 Cases

Das Schema der Cases Tabelle hat sich gegenüber der Studienarbeit nicht weiterentwickelt. In den Cases werden die Informationen zu den einzelnen Problemfällen gespeichert. Der Hauptpunkt der Cases ist, dass ein Techniker diese konsultieren kann, um zu sehen, was bereits vom System durchgeführt wurde. Somit muss er den Troubleshooting Prozess, auch bei einem Problem, dass nicht automatisch gelöst werden konnte, nicht komplett von vorne starten. Da für Netbox und die Speicherung der Devices eine PostgreSQL Datenbank benötigt wurde, wurden die Fälle ebenfalls in Postgres migriert. Während in der Studienarbeit die Cases noch als Überprüfung der Arbeitsschritte während eines Workflows dienten, sind sie nun mehr eine Überprüfung nach der abgeschlossenen Arbeit.

case_nr: Eine zwölfstellige, zufällig generierte ID die dazu dient, einen spezifischen Fall in der Datenbank wiederzufinden.

Event: Hier wird das Event gespeichert, welches den Troubleshooting Fall ausgelöst hat, dieser könnte zum Beispiel ein Interface down Event sein.

Description: Eine kurze Beschreibung für das Problem, dies ist vor allem dann wichtig, wenn ein Case manuell erstellt wird.

Status: Status ist ein ENUM, der beschreibt wie weit das Problem bereits behoben wurde. Es gibt „new“ für alle neu erstellten Fälle, „solution_deployed“ für Fälle bei denen das System zurzeit versucht diese zu beheben, „technician_needed“ falls ein Fall nicht automatisch behoben werden konnte und es einen manuellen Eingriff benötigt, „technician_on_case“ falls zurzeit ein Techniker versucht dieses Problem zu lösen und schliesslich „resolved“ wenn ein Problem behoben wurde und der Fall abgeschlossen wurde.

Created: Die Zeit und das Datum, wann der Fall erstellt wurde.

last_updated: Der Zeitpunkt, an dem der Fall das letzte Mal bearbeitet wurde.

Technician: Falls ein Techniker manuell versucht diesen Fall zu beheben, wird dieser hier eingetragen.

Sender device: Der Hostname des Geräts, bei welchem das Problem aufgetreten ist, welches zur Erstellung eines Falls führte, wird hier gespeichert.

Solution: Eine Liste die alle Lösungsschritte beinhaltet welche zur Behebung des Problems vom System unternommen wurde. Diese könnte wie folgt aussehen.

- Interface down status on host R15 detected. Config for Interface `GigabitEthernet2.45` automatically changed from down to up
- Executed workflow:



- Case started in kafka event consumer: `correlate.aggregate`
- Waiting for 10 seconds to aggregate distinct events.
- Aggregation not successful: sending `syslog/\*/INTERFACE\_CHANGED/down` event to salt master.
- Got neighbor of R15 on interface: GigabitEthernet2.45
- Got neighbors of R14
- Checking connectivity from R11 to R14. Result: True
- Trying to apply shutdown to interface `GigabitEthernet2.45` on host R15.
- Trying to apply no shutdown to interface `GigabitEthernet2.45` on host R15.
- Ping from R15 to R14. Result: True
- Workflow finished. Case-ID: WnB0Vjyqk6z



3.10 Prozesse und Threads

Prozesse und Threads waren während dieser Bachelorarbeit die wohl grösste programmiertechnische Herausforderung und werden an mehreren Stellen eingesetzt.

3.10.1 Prozesse

Da OATS aus mehreren Python-Skripts besteht, die zur gleichen Zeit auf einem Server laufen, wurden alle diese Skripts durch Verwendung von Systemd daemonisiert. Damit laufen diese Programme im Hintergrund und der User muss sich nicht selbst darum kümmern.

Systemd-Prozesse

Die folgenden Prozesse sind aktiv, wenn OATS gestartet wird:

- **oats_telemetry:** der Streaming-Telemetry Client von OATS (selbst entwickelt), zuständig für das Aufrechterhalten der Netconf-Sessions über welche die Streaming-Telemetry Daten übertragen werden.
- **napalm-logs:** der Syslog Client von OATS, zuständig für das Verarbeiten der syslog-messages.
- **vflow:** der Netflow Client von OATS, zuständig für das Verarbeiten der Netflow-Daten.
- **oats_napalm_logs_client:** nimmt die Daten von napalm-logs entgegen und leitet sie an Kafka-Topics weiter.
- **oats_api:** der Flask-basierte OATS-API Prozess welcher eine REST-Schnittstelle für das Erzeugen von Events innerhalb von OATS dient.
- **oats-generic-kafka-streams:** Der generische Kafka-Streams Prozess von OATS. Wird zurzeit nur einmal ausgeführt, da dies so in der Konfiguration angegeben ist. Könnte aber auch je nach Konfiguration mehrmals ausgeführt werden.
- **syslog_ospf_nbr_down_consumer:** Kafka-Consumer für OSPF_NEIGHBOR_DOWN Events.
- **syslog_ospf_nbr_up_consumer:** Kafka-Consumer für OSPF_NEIGHBOR_UP Events.
- **syslog_ifchanged_consumer:** Kafka-Consumer für INTERFACE_CHANGED Events.
- **out-discards-events_consumer:** Kafka-Consumer für OUT_DISCARD Events.
- **oats-api_consumer:** Kafka-Consumer für API Events.
- **oats-netflow_consumer:** Kafka-Consumer für Netflow-Daten.
- **oats-generic-kafka-streams_consumer:** Kafka-Consumer für Streaming-Telemetry Daten.

Evaluation der Auslastung

Die meisten dieser Prozesse sind nur aktiv, wenn Events auftreten (alles was mit Syslog/napalm-logs zu tun hat.). Diese sind zu >99% inaktiv.

Am meisten ausgelastet sind diejenigen Prozesse, welche mit Streaming-Telemetry Daten oder Netflow-Daten arbeiten. Aber auch diese sind beim derzeitigen Lab-Netzwerk noch sehr wenig ausgelastet, da das Datenvolumen noch klein ist. Will man dies in einem richtigen Netzwerk mit einem grossen Datenvolumen aufsetzen bieten sich hier 2 Möglichkeiten:

- **Leistungsfähige Hardware:** Ist die einfachste Lösung und auch einfach aufzusetzen. Allerdings wird man auch damit irgendwann an seine Grenzen stossen.
- **Verteiltes Deployment:** In der Theorie ist es möglich, OATS als komplett verteiltes System aufzusetzen. So könnten verschiedene Deployments auf unterschiedlichen Servern unterschiedliche Daten verarbeiten. Dies ist dank der anpassungsfähigen Konfiguration und dank den einzelnen skalierfähigen Komponenten (Kafka, InfluxDB) möglich.



3.10.2 Threads

Threading wird an mehreren Stellen benutzt:

`oats_telemetry.py`

Die Anzahl Threads, die hier gestartet werden, unterscheiden sich je nach Grösse des Konfigurationsfiles. In der BA wurde OATS für 2 Hosts und 1 Subscription konfiguriert. Somit wird für jeden Host ein Thread für die Netconf-Session (**`netconf_telemetry.py`**) gestartet und für jede Subscription wird ein Kafka-Streams Client (**`oats-generic-kafka-streams.jar`**) gestartet. Zusätzlich wird in **`netconf_telemetry.py`** pro Subscription wieder ein Thread gestartet.

Alles zusammengezählt bringt dies die Threadanzahl auf 5.

Bei einem sehr grossen Konfigurationsfile (z.B. >10 Hosts & >10 Subscriptions) wächst die Anzahl Threads stark, da für jeden Host pro Subscription ein Thread gestartet wird. Bei 10 Hosts und 10 Subscriptions wären dies $10 + 10 \cdot 10 = 110$ Threads. Bei solch grossen Konfigurationsfiles würde sich das Verteilen von OATS auf verschiedene Server anbieten.

`tshoot.py`

Hier wird an mehreren Stellen Threading eingesetzt, um den Erfolg eines Troubleshooting-Workflows zu überprüfen. Dabei wird asynchron auf ein Salt-Event gewartet, welches den Erfolg des Workflows bestätigt.

Rein von der Auslastung des Systems können diese Threads vernachlässigt werden, da diese Anzahl auch bei grosser Last nie grösser als 10 werden sollte.

Kafka-Consumer

Die verschiedenen Kafka-Consumer, die im obigen Kapitel «Systemd-Prozesse» bereits vorgestellt wurden, benutzen, zur Korrelation von Events, Threading. Die Korrelations-Funktionen die beim Threading benutzt werden befinden sich im File **`oats_correlate.py`**.

Dabei werden Events für eine gewisse Zeit entweder aggregiert oder komprimiert, indem jedes neue Event in diesem Zeitraum einen neuen Thread startet und dann einen Zähler inkrementiert. So kann die Anzahl der komprimierten/aggregierten Events gezählt werden kann.

Dies kann für eine kurze Zeit zu einer grossen Anzahl von Threads führen, wenn in einem kurzen Zeitabstand sehr viele Events auf dem OATS-Server eintreffen. Da die meisten Threads aber nur sehr kurz leben, weil sie nur den Zähler inkrementieren müssen, sehen wir hier für die Zukunft keine Probleme.

3.11 Grössen und Leistung

Während des Projekts wurden keine grösseren Benchmark-Tests o.Ä. durchgeführt. Es wurden aber manuell viele Events generiert und das Verhalten von OATS beobachtet. Die einzigen Probleme wurden dabei innerhalb von Salt entdeckt.

3.11.1 Salt-Troubleshooting-Workflows

Bei vielen Events innerhalb von kurzer Zeit entstehen Probleme innerhalb von Salt, da dann mehrere Prozesse gleichzeitig versuchen auf die gleichen Netzwerkgeräte per SSH zuzugreifen und evtl. auch versuchen neue Konfigurationen zu laden. Dies führt zu Konflikten und manche dieser Prozesse werfen Fehlermeldungen und brechen ab. Dies stellt aber kein Problem dar, da im Worst-Case keine Konfiguration auf das Gerät geladen wird (durch napalm sichergestellt, da `load_config` atomar ist). Auch der Salt-master ist davon unbeeinflusst, da er die Exceptions zwar anzeigt (im Debug-Modus), aber danach schluckt.



3.11.2 Laden von Konfigurationen

Ein weiteres Problem von Salt in Bezug auf Leistung, sind die langsamen Befehle zum Laden von Konfigurationen auf Netzwerkgeräte (dauert ca. 45 Sekunden). Es ist noch unklar woran das genau liegt, aber wir haben die Vermutung, dass dies über napalm-ios (das napalm-cisco Modul) nicht schneller möglich ist, da dieses Verhalten während der gesamten Projektdauer gleichgeblieben ist.

Dies stellt bei einem Workflow ein Problem dar, da in diesem nicht direkt versucht wird das Problem zu beheben, sondern nur für eine gewisse Zeit der Verkehr beim betroffenen Gerät gedrosselt wird.

Im Idealfall würden zwischen dem Entdecken des Problems bis zum Drosseln des Verkehrs höchstens ein paar wenige Sekunden verstreichen. Dies ist auch der Fall, bis das Event bei Salt ankommt und dort versucht wird eine Konfiguration zu laden, welche den Verkehr drosselt, was ca. 45 Sekunden dauert.

3.11.3 Fazit

Es hat sich gezeigt, dass wenn es einen Schwachpunkt im System beim Skalieren von OATS gibt, dann ist es Salt. Während der Studienarbeit war Salt noch gut für das automatische Troubleshooting geeignet, stiess aber im Verlaufe der Bachelorarbeit an seine Grenzen.

Vor allem das langsame Laden von Konfiguration ist ein Problem welches, je nach Anwendungsfall, nicht ignoriert werden kann. Als einer der ersten Schritte beim Weiterentwickeln von OATS wäre zu untersuchen, wieso das Laden derart langsam ist. Kommt man zum Schluss, dass dies mit Salt nicht anders möglich ist, so müsste Salt durch etwas ersetzt werden, was diese Befehle in weniger als 5 Sekunden durchführen kann.



4. Anleitungen

Änderungsgeschichte

Datum	Version	Änderung	Autor
05.12.2017	1.0	Erstellen des Dokuments	rj
11.12.2017	1.1	Ergänzung Installationsanleitung Schreiben Benutzungsanleitung	nv
12.12.2017	1.2	Schreiben Development	nv
04.06.2018	2.0	Übernahme Dokument in BA	nv
05.06.2018	2.1	Überarbeitung vor Abgabe	rj



4.1 Einführung

4.1.1.1 Zweck

In diesem Dokument wird beschrieben, wie OATS installiert benutzt und konfiguriert werden kann. Im letzten Kapitel wird zudem beschrieben, wie OATS erweitert werden kann.

4.1.1.2 Gültigkeitsbereich

Das Dokument ist gültig während der Entwicklung des automatisierten Troubleshooting Systems OATS.



4.2 Installationsanleitung

Anmerkung: Es wird empfohlen im Zusammenhang mit OATS als Root auf dem Server zu arbeiten, da das darunterliegende Salt-System Root-Rechte voraussetzt.

Die folgenden Installationsanleitungen sind für Linux Ubuntu 16.04 ausgelegt. Für Anleitung zu anderen Betriebssystemen, verweisen wir auf die offiziellen Anleitungen der Hersteller.

4.2.1 Python

Als Programmiersprache und zur Ausführung verschiedener Module wird Python 2.7.14 benötigt, diese ist zum Downloaden auf der offiziellen Python Seite verfügbar.

<https://www.python.org/downloads/>

Eine mögliche Art zum Installieren unter Linux ist:

```
sudo apt-get update  
sudo apt-get install -y python-pip python-dev build-essential
```

Python & OATS haben noch einige dependencies die möglicherweise bereits vorhanden sind. Es wird trotzdem empfohlen diese noch zu installieren:

```
sudo apt-get install libssl-dev libffi-dev python-cffi
```

4.2.2 Pip

Pip ist ein Python Package Manager welcher benötigt wird um weitere Module zu installieren, in der oben erwähnten Python Version ist pip bereits enthalten, ein separater Download ist verfügbar unter:

<https://pip.pypa.io/en/stable/installing/>

Eine mögliche Art der Installation ist:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py  
sudo python get-pip.py
```

Da die in OATS eingesetzte Version von napalm interne Funktionen der pip-API benutzt, welche mit der pip-Version 10 entfernt wurden, ist es wichtig, dass eine pip-Version installiert wird, die kleiner als 10 ist:

```
sudo pip install --upgrade pip==9.0.1
```

4.2.3 Apt Packages

Git

Git wird benötigt um das OATS Github-Repo zu klonen:

```
sudo apt-get install git-core
```

Systemd

Systemd wird benötigt, damit die einzelnen OATS-Komponenten als Daemons auf dem Server aufgesetzt werden können:

```
sudo apt-get install systemd
```

Java 8

Java 8 wird von Kafka benötigt. Zuerst muss das Repository hinzugefügt werden:

```
sudo add-apt-repository -y ppa:webupd8team/java && sudo apt-get update &&  
sudo apt-get -y upgrade
```



Obige Befehle müssen mehrmals per CLI bestätigt werden.

Zum Installieren von Java 8:

```
sudo apt install -y oracle-java8-installer
```

4.2.4 Kafka

Kafka dient als Message-Queue innerhalb von OATS. Die aktuellste Version befindet sich unter:

<https://kafka.apache.org/quickstart>.

Die während der Entwicklung eingesetzte Version kann folgendermassen installiert werden:

```
wget http://mirror.switch.ch/mirror/apache/dist/kafka/1.1.0/kafka_2.11-1.1.0.tgz
tar -xzf kafka_2.11-1.1.0.tgz
```

Es empfiehlt sich, den Zookeeper- und den Kafka-Prozess direkt zu starten:

```
./kafka_2.11-1.1.0/bin/zookeeper-server-start.sh -daemon kafka_2.11-1.1.0/config/zookeeper.properties
./kafka_2.11-1.1.0/bin/kafka-server-start.sh -daemon kafka_2.11-1.1.0/config/server.properties
```

4.2.5 Influx

Influx dient als Datenbank für die Timeseries-Daten, die komplette Installationsanleitung ist zu finden auf <https://docs.influxdata.com/influxdb/v1.5/>.

Wie anfangs bereits erwähnt, folgt hier nur die Installationsanleitung für Ubuntu 16.04. Zuerst muss das InfluxData Repository hinzugefügt werden:

```
curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -
source /etc/lsb-release
echo "deb https://repos.influxdata.com/${DISTRIB_ID,,} ${DISTRIB_CODENAME} stable" | sudo tee /etc/apt/sources.list.d/influxdb.list
```

Danach muss der Influxdb Service installiert und gestartet werden.

```
sudo apt-get update && sudo apt-get install Influxdb
sudo service influxdb start
```

Um die Konfiguration für Influxdb zu laden gibt es zwei unterschiedlicher Wege.

Der Prozess kann mit folgendem Befehl zu dem richtigen Konfigurations File zugewiesen werden.

```
influxd -config /etc/influxdb/influxdb.conf
```

Die Umgebungsvariable kann mit folgendem Befehl auf das Konfigurationsfile gesetzt werden.

Danach den Prozess neu starten.

```
echo $INFLUXDB_CONFIG_PATH
/etc/influxdb/influxdb.conf
```

```
influx
```

4.2.6 Postgres

PostgreSQL kann entweder über den Package Manager des Betriebssystems installiert werden oder direkt vom offiziellen Repository.

```
apt-get update
apt-get install -y postgresql libpq-dev
```



Danach muss der Service gestartet werden und die Erlaubnis erhalten, beim Boot ausgeführt zu werden.

```
systemctl start postgresql
systemctl enable postgresql
```

Danach muss die Datenbank erstellt werden, sowie ein User und das Passwort hinzugefügt werden.

```
sudo -u postgres psql
psql (9.4.5)
Type "help" for help.
```

```
postgres=# CREATE DATABASE netbox;
CREATE DATABASE
postgres=# CREATE USER netbox WITH PASSWORD 'INSERT_PASSWORD_HERE';
CREATE ROLE
postgres=# GRANT ALL PRIVILEGES ON DATABASE netbox TO netbox;
GRANT
postgres=# \q
```

Danach muss noch die Cases Table erstellt werden. Diese ist in unserem Repository unter `OATS/saltstack/database/data/templates/casetable.sql` zu finden. Der Code dieses Files kann direkt in die PSQL Konsole eingefügt werden und der Import erfolgt. Ist diese Tabelle importiert, ist die PostgreSQL Datenbank bereits operationsfähig. Es gibt jedoch noch weitere Konfigurationsschritte, die unternommen werden können. Vor allem in Bezug auf die Sicherheit. Hierzu verweisen wir auf die offizielle Dokumentation.

<https://www.postgresql.org/docs/10/static/index.html>

4.2.7 Netbox

Netbox dient als Zwischenstück für die Geräte-Daten und das System, sowie als Visualisierung der Daten für die Nutzer. Die komplette Installationsanleitung ist zu finden auf

<https://netbox.readthedocs.io/en/latest/installation/netbox/>.

Zuerst muss sichergestellt werden, dass die Python Tools installiert sind.

```
sudo apt-get install -y python2.7 python-dev python-setuptools build-essential libxml2-dev libxslt1-dev libffi-dev graphviz libpq-dev libssl-dev zlib1g-dev
sudo easy_install pip
```

Am einfachsten klonet man als nächstes das offizielle Repository. Zuerst muss ein Ordner erstellt werden und in dessen Verzeichnis gewechselt werden, danach folgt das eigentliche Klonen:

```
mkdir -p /opt/netbox/ && cd /opt/netbox/
```

```
# git clone -b master https://github.com/digitalocean/netbox.git .
```

Da man mit dem Befehl von oben bereits in das Repository gewechselt hat, kann man nun einfach die requirements installieren.

```
pip install -r requirements.txt
```

Grundsätzlich ist Netbox so installiert. Jedoch muss noch einiges an Konfiguration durchgeführt werden. Genauer dazu steht im Kapitel 4.3 (Konfigurationsanleitung).



4.2.8 OATS

Nun kann OATS mitsamt den dependencies installiert werden. Dazu muss lediglich das OATS Github Repository geklont und folgende Schritte durchgeführt werden

```
git clone https://github.com/nvinzens/OATS  
sudo pip install -r requirements.txt  
sudo ./oats_bootstrap.sh
```

Die Datei requirements.txt enthält alle Python-Dependencies von OATS (napalm, Salt, napalm-logs, etc).

4.2.9 Vagrant

Unter /OATS/vagrant ist ein Vagrantfile vorhanden, über welches eine Ubuntu 16.04 Virtual-Box installiert werden kann. Bei der Installation werden direkt alle OATS-dependencies und OATS selbst installiert und aufgesetzt.

Damit diese Box Sinn macht muss diese noch ein Netz von Cisco-Geräten eingefügt und die Konfiguration darauf angepasst werden



4.3 Konfigurationsanleitung

Gewisse Teile von OATS müssen noch mittels Konfigurationsdateien an die Umgebung angepasst werden.

4.3.1 OATS

OATS verfügt über eine zentrale Konfigurationsdatei, welche sich unter `/etc/oats` befindet, mit welcher Streaming-Telemetry und Event-Korrelation eingerichtet werden können. Die OATS Konfiguration ist im YAML-Format formatiert und sieht folgendermassen aus:

```
hosts:
- host: 10.20.1.22
  port: 830
  username: ins
  password: ins@lab
- host: 10.20.1.21
  port: 830
  username: user
  password: password

subscriptions:
- subscription:
  xpath: /if:interfaces-state/interface/statistics/out-discards
  period: 1000
  kafka_publish_topic: interfaces-out-discards
  kafka_streams_eval: true
  kafka_streams_jar_location: /usr/local/bin/oats/oats-generic-kafka-
streams.jar
  event_threshold_data:
  - value: 100000
    operator: greater_than
    kafka_event_topic: out-discards-events
    data_xpaths:
    - root_xpath: /notification/push-update/datastore-contents-
xml/interfaces-state/interface
      name_xpath: /name
      data_xpath: /statistics/out-discards
    event: streaming-telemetry/*/out-discard-event
    correlate_event: true
    correlate:
    - function: compress
      correlation_time: 10
```

In den nachfolgenden Unterkapiteln werden die einzelnen Abschnitte der Konfiguration erklärt.

Hosts

Im `hosts`-Abschnitt wird angegeben zu welchen Netzwerkhosts eine Netconf-Session aufgemacht werden soll, damit über diese Streaming-Telemetry-Daten übertragen werden können. Die einzelnen Felder werden für folgende Informationen benötigt:

- **host:** Die IP der Hosts zu denen die Netconf-Session aufgemacht werden soll.
- **port:** Die Portnummer die für die Netconf-Session verwendet werden soll. Standardmässig wird für Netconf Port 830 verwendet.
- **username:** Der Benutzername mit welchem man sich auf dem Gerät einloggen kann.
- **password:** das Passwort des Benutzers.



Subscriptions

Der *subscriptions*-Abschnitt gibt an:

- Auf welches Yang-Modell subscribed werden soll. Unter <https://github.com/YangModels/yang/tree/master/vendor/cisco> ist einsehbar, welche Modelle auf Cisco Geräten verfügbar sind.
- Ob die Daten in den Modellen im eingebauten Kafka-Streams Client evaluiert werden sollen.
- Was mit möglichen Events bei der Evaluation geschehen soll.
- Ob die Events korreliert werden sollen.

Nachfolgend werden die einzelnen Felder erklärt:

- **subscription**: enthält alle Informationen die eine Subscription auf ein YANG-Modell benötigt. Für jeden oben angegeben host wird diese Subscription über die bestehende Netconf-Session eröffnet. Es ist denkbar hier viele Subscriptions anzugeben, im Projekt wurde bisher mit nur einer gearbeitet.
 - **xpath**: Der xpath zeigt an, auf welches Modell subscribed werden soll. Mit einem xpath lässt sich elegant angeben, welche Daten innerhalb eines Yang-Modells per subscription geliefert werden sollen.
 - **period**: Gibt an in welchem regelmässigen Intervall (in Millisekunden) die Daten geliefert werden sollen
 - **kafka_publish_topic**: Gibt an in welchem Kafka-Topic die Daten eingereicht werden sollen. Kann grundsätzlich frei gewählt werden und es wäre auch denkbar die Daten in ein bereits existierendes Topic einzuschleusen, je nach dem was mit den Daten geschehen soll.
 - **kafka_streams_eval**: *true/false*, gibt an, ob die Daten im oben angegeben Topic mittels Kafka-Streams Client evaluiert werden sollen.
 - **kafka_streams_jar_location (optional)**: Zeigt auf den Ort an dem das Kafka-Streams jar-File abgelegt ist. Ist standardmässig unter */usr/local/bin/oats/kafka-streams/*. Muss nur angepasst werden, falls ein anderer Client eingesetzt werden soll als der, der während der Bachelorarbeit entwickelt wurde.
 - **event_threshold_data (optional)**: Enthält die Daten, die vom Kafka-Streams Client benötigt werden. Dieses Feld und die darunterliegenden Elemente (*bis data_xpath*) müssen angegeben werden, falls *kafka_streams_eval* auf *true* gesetzt ist.
 - **value (optional)**: Der Schwellenwert der mittels des Vergleichs-Operators evaluiert wird.
 - **operator (optional)**: Der Vergleichsoperator der eingesetzt werden soll. Bei *greater_than* wird so ein Event ausgelöst, sobald die Differenz des aktuellsten Werts mit dem vorherigen Wert, den Schwellenwert in *value* überschreitet.
 - **kafka_event_topic (optional)**: Das Topic, in welches die oben beschriebenen Events eingereicht werden sollen.
 - **data_xpaths (optional)**: Gibt an, wo sich die zu evaluierenden Werte der Streaming-Telemetry Daten befinden. Setzt genaues Wissen über die Streaming-Telemetry Daten voraus.
 - **root_xpath (optional)**: Der Root-Xpath, der zu den gewünschten Werten innerhalb der Daten führt.
 - **name_xpath (optional)**: Ist relativ zum *root_xpath* und zeigt an, wo sich der Name des zu evaluierenden Wertes befindet (z.B. der Name des Interfaces von welchem die Daten des Hosts stammen).
 - **data_xpath (optional)**: Ist relativ zum *root_xpath* und zeigt an, wo sich der Wert befindet, der mittels des Vergleichsoperators evaluiert werden soll. Beim Operator *greater_than* wird so ein Event ausgelöst, sobald die Differenz des aktuellsten Werts



mit dem vorherigen Wert, den in *value* angegeben Schwellenwert überschreitet. Das Event enthält den aktuellsten Wert und dessen Namen.

- **event (optional):** Enthält den Event-Namen der intern von OATS verwendet wird. Wird benötigt um z.B. innerhalb von Salt einen Troubleshooting-Workflow anzustossen (Der Salt Reactor horcht auf diesen Namen). Wird zudem in der InfluxDB abgelegt um die Events auseinanderhalten zu können.
- **correlate_event:** true/false, gibt an ob die Events noch zusätzlich korreliert werden sollen.
- **correlate (optional):** Enthält die Informationen, die zum Korrelieren der Events benötigt werden. Dieses Feld und die darunterliegenden Felder müssen angegeben werden, wenn *correlate_event* auf *true* gesetzt ist.
- **function (optional):** Die Korrelationsfunktion die benutzt werden soll. Bis zum Ende der Bachelorarbeit wurden *aggregate_identical*, *aggregate_distinct* und *compress* implementiert.
- **correlation_time (optional):** Der Zeitraum, für den korreliert werden soll, in Sekunden.

Minimale Konfiguration

Das Beispiel oben zeigt eine maximale Konfiguration mit zwei Hosts und einer subscription an. Der Vollständigkeit halber folgt noch ein zweites Beispiel, welches nur die Streaming-Telemetry Subscriptions zu den Hosts aufbaut und die Daten in das angegeben Kafka-Topic schreibt (keine Kafka-Streams Evaluation und keine Korrelation).

hosts:

- **host:** 10.20.1.22
port: 830
username: ins
password: ins@lab
- **host:** 10.20.1.21
port: 830
username: user
password: password

subscriptions:

- **subscription:**
xpath: /if:interfaces-state/interface/statistics/out-discards
period: 1000
kafka_publish_topic: interfaces-out-discards
kafka_streams_eval: false
correlate_event: false



4.3.2 Napalm-Logs

Die napalm-logs Konfigurationsdatei befindet sich unter (logs ist die Konfigurationsdatei):

```
/etc/napalm/logs
```

Darin werden werden die Listener- sowie die Publisher-Adresse spezifiziert.

Die initiale Konfiguration sollte durch das Bootstrap-Skript bereits durchgeführt worden sein. Was möglicherweise angepasst werden muss, sind die IP-Adressen. Es ist wichtig, dass die Port-Nummern und der Transport **nicht** geändert werden.

```
address: 10.20.1.10
port: 33333
publish_address: 10.20.1.10
publish_port: 49017
transport: zmq
disable_security: true
log_level: info
publisher:
  zmq:
    send_unknown: true
log_file: /etc/napalm/log
```

`address` und `publish_address` geben an, unter welcher IP-Adresse auf syslog-messages gehorcht wird und unter welcher IP die resultierenden napalm-logs-messages veröffentlicht werden. Diese müssen mit der IP-Adresse des Servers auf dem OATS läuft übereinstimmen (in der SA war dies 10.20.1.10).

`port` gibt an, auf welchem Port auf syslog-messages gehorcht wird. Er kann frei gewählt werden, es muss aber sichergestellt sein, dass er nicht von einem anderen Prozess belegt ist. In der SA wurde Port 33333 gewählt.

`publish_port` gibt an, auf welchem Port napalm-logs-messages veröffentlicht werden. Dieser muss auf 49017 gesetzt sein, da der dazugehörige Client (oatsclient.py) standardmässig die messages von diesem Port erwartet.

Der Rest der Datei sollte so belassen werden.

4.3.3 Salt-Proxies

Salt-Proxies dienen der Kommunikation mit den Netzwerkgeräten. Dabei wird für jedes Gerät, welches per OATS angesteuert werden können soll, ein Proxy erstellt.

Die Proxy-Konfigurationsdateien befinden sich unter:

```
/srv/saltstack/pillar
```

In top.sls müssen alle Proxies angegeben werden.

Die Proxy-Konfigurationsdateien haben folgende Form (Beispiel: R11 aus der SA):

```
proxy:
  proxytype: napalm
  driver: ios
  host: 10.20.1.11
  username: ***
  passwd: ***
  provider: napalm
  optional_args:
    secret: ***
```

Angepasst werden müssen folgende Felder:

`host`: Die IP-Adresse des Netzwerkgeräts

`username`: Der Username mit dem man sich auf dem Gerät einloggen kann



passwd: Das Passwort für den User

secret: Das Enable-Passwort des Routers (optional)

Die restlichen Felder sollten belassen werden.

Es ist zu empfehlen das Hosts-file (/etc/hosts) des Servers auf dem OATS läuft mit den Netzwerkgerät-Hostnamen zu ergänzen, z.B.:

```
10.20.1.11 R11
10.20.1.12 R12
10.20.1.13 R13
```

4.3.4 Netbox

Wie bereits in der Installationsanleitung zu Netbox erwähnt folgt hier nun die Konfiguration, ohne die Netbox nicht ausführbar ist.

Zuerst muss man ins richtige Verzeichnis wechseln und eine Kopie der Konfigurationsvorlage erstellen.

```
cd /opt/netbox/netbox/netbox/
cp configuration.example.py configuration.py
```

Danach muss diese Datei mit einem Editor geöffnet werden um die Informationen darin zu editieren. Zuerst muss unter dem Punkt ALLOWED_HOSTS der Hostname definiert werden, mit welche der Server erreicht werden kann.

```
ALLOWED_HOSTS = ['netbox.example.com', '192.0.2.123']
```

Danach müssen im gleichen Dokument die Datenbank Informationen so gesetzt werden, wie die in der PostgreSQL Datenbank.

```
DATABASE = {
    'NAME': 'netbox',          # Database name
    'USER': 'netbox',         # PostgreSQL username
    'PASSWORD': 'INSERT_PASSWORD_HERE', # PostgreSQL password
    'HOST': 'localhost',      # Database server
    'PORT': '',               # Database port (leave blank for
default)
}
```

Darauf folgt das Einfügen eines Secret Keys. Dieser muss gemäss Dokumentation mindestens 50 alphanummerische Zeichen enthalten. Dazu gibt es unter

/opt/netbox/netbox/generate_secret_key.py ein Generator Skript.

Damit Netbox läuft muss noch das Schema in der Datenbank installiert werden, dies kann man mit:

```
cd /opt/netbox/netbox/
python manage.py migrate
```

Darauf folgt das Erstellen eines Super Users mit welchem man auf Netbox zugreifen kann.

```
python3 manage.py createsuperuser
Username: admin
Email address: admin@example.com
Password:
Password (again):
Superuser created successfully.
```

Als letztes müssen noch die statischen Files gesammelt werden.

```
python3 manage.py collectstatic --no-input
```

Nun ist die Installation und Konfiguration von netbox beendet. Man kann sie mit folgendem Befehl testen.

```
python manage.py runserver 0.0.0.0:8000 --insecure
```



4.3.5 Netzwerkgeräte

Alle Geräte die Daten an OATS schicken sollen müssen wie folgt konfiguriert werden:

Privilege

```
router(config)#username <user> privilege 15
```

user: Der User, der in den Proxies angegeben wurde. So wird sichergestellt, dass der User über alle Rechte verfügt, die er benötigt um via napalm Befehle auszuführen.

Syslogging

```
router(config)#logging host <OATS-IP> transport udp port <napalm-logs-port>
```

OATS-IP: Die IP, welche in der napalm-logs Konfigurationsdatei als Listener-IP angegeben wurde (10.20.1.10 in der SA).

napalm-logs-port: Der Port der in der napalm-logs Konfigurationsdatei als Listener-Port angegeben wurde (33333 in der SA).

```
router(config)#logging trap <level>
```

level: Der Syslog-Logging-Level (empfohlen: 5)

Hostnames

Damit die bestehenden Workflows funktionieren, muss auf jedem Gerät konfiguriert werden, welche Hostnamen die umliegenden Geräte verfügen und über welche IP diese erreichbar sind:

```
router(config)#ip host <hostname> <IP-address>
```

hostname: Hostname des Routers, z.B. R11

IP-address: IP-Adresse des Routers, z.B. 172.16.12.1

Dies muss für jeden umliegenden Router auf jedem Router wiederholt werden.

Streaming-Telemetry

Damit Streaming-Telemetry funktioniert, muss netconf-yang aktiviert werden. Genauere

Informationen findet man unter https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/166/b_166_programmability_cg/model_driven_telemetry.html

Während dem Development mussten die Router im Lab-Netz noch auf neuere Version geupdatet werden. Wir empfehlen daher das Verwenden einer Version von IOS-XE >= 16.8.1.

Netconf-yang kann wie folgt aktiviert werden:

```
sw01#netconf-yang
```



Netflow

Die Konfiguration von Netflow auf IOS-XE ist relativ aufwändig. Deswegen folgen einige Screenshots der Switch-Konfiguration, wie sie in OATS verwendet wurde:

Als erstes müssen die Records konfiguriert werden, jeweils einer für input und einer für output. Diese sind dafür zuständig, welche Informationen in den einzelnen Netflow-Meldungen enthalten sind.

Als Zweites folgt der Flow-Exporter, welcher angibt, wohin die Netflow-Meldungen übertragen werden sollen.

```
flow record oats-input
description IPv4 NetFlow
match ipv4 source address
match ipv4 destination address
match transport source-port
match transport destination-port
match ipv4 protocol
match interface input
match ipv4 tos
match flow direction
collect interface output
collect counter bytes long
collect counter packets long
collect transport tcp flags
collect timestamp absolute first
collect timestamp absolute last
!
!
flow record oats-output
description IPv4 NetFlow
match ipv4 source address
match ipv4 destination address
match transport source-port
match transport destination-port
match ipv4 protocol
match interface output
match ipv4 tos
match flow direction
collect interface input
collect counter bytes long
collect counter packets long
collect transport tcp flags
collect timestamp absolute first
collect timestamp absolute last
```

Abbildung 57: Flow-Records

```
flow exporter ExportTest
destination 10.20.1.10 vrf INS
transport udp 2055
!
!
flow monitor oats-mon-output
description IPv4 egress exports
exporter ExportTest
cache timeout active 60
record oats-output
!
!
flow monitor oats-mon-input
description IPv4 ingress exports
exporter ExportTest
cache timeout active 60
record oats-input
```

Abbildung 56: Flow Exporter & Monitore

```
interface GigabitEthernet2/0/2
ip flow monitor oats-mon-input input
ip flow monitor oats-mon-output output
!
interface GigabitEthernet2/0/3
ip flow monitor oats-mon-input input
ip flow monitor oats-mon-output output
!
interface GigabitEthernet2/0/4
switchport access vlan 22
switchport mode access
ip flow monitor oats-mon-input input
ip flow monitor oats-mon-output output
!
interface GigabitEthernet2/0/5
ip flow monitor oats-mon-input input
ip flow monitor oats-mon-output output
```

Abbildung 55: Flow-Monitore auf den Interfaces

Danach müssen Flow-Monitore definiert werden, auch hier wieder jeweils einer für input und output. Die Monitore vereinen die Records und die Exporter und können dann auf Interfaces angewendet werden, wie man im letzten Screenshot sieht.



4.4 Benutzungsanleitung

Folgende Anleitung setzt voraus, dass alle Komponenten korrekt installiert wurden und sich alle relevanten Dateien am richtigen Ort befinden.

Da Salt voraussetzt, dass der Salt-prozess über Root Rechte verfügt, müssen auch nachfolgenden Schritte mit Root-Rechten ausgeführt werden. Dazu dient folgender Befehl:

```
~$sudo su
```

4.4.1 OATS

Das Deployment wurde während der Bachelorarbeit vereinfacht, weswegen nur noch ein Befehl zum Starten aller OATS-Scripts als Daemons ausgeführt werden muss:

```
~#systemctl start oats.target
```

Danach kann mit `ps -aux | grep oats` kontrolliert werden, ob alle Daemons vorhanden sind.

Das einzige, was nicht über *oats.target* verwaltet wird, sind die Salt-prozesse, welche manuell verwaltet werden müssen.

Sollte etwas nicht funktionieren, befindet sich unter `/var/log/oats/logs` eine Log-Datei in welcher nachverfolgt werden kann, was schiefgelaufen ist.

4.4.2 Salt Environment

Folgende Abschnitte dienen nur zum Aufzeigen, wie alle Salt spezifischen Prozesse gestartet werden können. Für ein tieferes Verständnis zu Salt sollte <https://docs.saltstack.com> konsultiert werden. Die Urheber dieses Dokuments empfehlen zusätzlich folgendes Tutorial durchzuführen:

<https://docs.saltstack.com/en/latest/topics/tutorials/walkthrough.html>

Salt Master

Der Salt-master dient als zentrale Sammelstelle für Events und führt als Reaktion darauf Workflows aus. Er übernimmt auch die Kommunikation mit den Salt-proxies und -minions. Zum Starten des Salt masters als Daemon muss folgender Befehl ausgeführt werden (optional: Debug-Modus):

```
# salt-master -d
```

bzw.

```
# salt-master -l debug
```

Salt Minion

Für OATS wird nur ein minion benötigt, nämlich derjenige der auf dem master selbst (dient zur Konfiguration des masters). Folgender Befehl startet den minion als Daemon:

```
# salt-minion -d
```

An diesem Punkt lohnt es sich zu überprüfen, ob alles korrekt funktioniert. Folgender Befehl testet, ob master und master-minion miteinander kommunizieren können:

```
# salt master test.ping
```

Dies sollte folgenden output generieren:

```
master:
```

```
True
```



Salt Napalm-Proxies

Nun müssen noch die Salt napalm-proxies gestartet werden. Die proxies dienen zur Kommunikation mit den Netzwerkgeräten und benutzen die in Salt integrierte napalm-API. Die proxies können mit folgendem Befehl als Daemon gestartet werden (wobei <ID> mit dem jeweiligen Namen des Proxies ersetzt werden muss). Die Namen der proxies müssen mit denen unter /srv/saltstack/pillar/übereinstimmen:

```
# salt-proxy --proxyid <ID> -d
```

Die in der Bachelorarbeit benutzten proxies waren R11-R16 und sw01/sw02. Mittels test.ping kann nun wieder getestet werden, ob die proxies hochgefahren wurden:

```
# salt '*' test.ping
```

Dies sollte folgenden output generieren (Reihenfolge kann variieren):

```
R16:
    True
SW02:
    True
master:
    True
R14:
    True
R12:
    True
SW01:
    True
R15:
    True
R11:
    True
R13:
    True
master:
    True
```

4.4.3 Netbox

Sobald man Netbox installiert und konfiguriert hat, kann man auf die IP Adresse mit dem definierten Port navigieren. Dort findet man bereits die graphische Benutzeroberfläche von Netbox. Da hier es sich hier um eine komplette Benutzeroberfläche handelt, verweisen wir bloss auf die Dokumentation von Netbox. <https://netbox.readthedocs.io/en/latest/>

4.4.4 PostgreSQL

Auf die PostgreSQL Datenbank kann man über die PSQL Konsole zugreifen.

```
# PSQL
>
```

Von dort aus erfolgen die Standard Kommandozeilen Befehle, wie sie es für eine SQL Datenbank üblich sind. Eine komplette Dokumentation findet man unter der PostgreSQL Dokumentation <https://www.postgresql.org/docs/10/static/sql-commands.html>.

Eine weitere Option ist die Verwendung einer graphischen Oberfläche wie etwa pgAdmin. <https://www.pgadmin.org/>. Zur Verwendung dieser verweisen wir auf die offizielle Dokumentation, da sie nicht Teil von OATS und dieser Arbeit ist.



4.4.5 Influx

Ist die Influx Datenbank erfolgreich installiert kann man einfach per Influx Konsole darauf zugreifen.

```
influx
Connected to http://localhost:8086 version 1.5.2
InfluxDB shell version: 1.5.2
>
```

Diese Art die Konsole zu öffnen liefert einem die Zeitangaben

Um zu sehen welche Datenbanken existieren kann man den Befehl SHOW DATABASES eingeben, dieser liefert einem eine Liste der existierenden Datenbanken.

```
> show databases
name: databases
name
----
_internal
timedb
```

Eine neue erstellen kann man einfach mit CREATE DATABASE DATABASENAME. Um mit der erstellten Datenbank zu interagieren, muss man zudem noch den USE DATABASENAME eingeben.

```
> create database timedb
```

```
> use timedb
Using database timedb
```

Um eine Tabelle oder unter Influx genannt Measurement abzufragen kann man SQL SELECT...FROM... Statement eingeben wie etwa.

```
> select * from oats_timeseries_netflow limit 1
name: oats_timeseries_netflow
```

Dieser liefert einem den ersten Eintrag in der Measurement Tabelle „oats_timeseries_netflow“.

4.4.6 API

Eine genaue Dokumentation findet man auf dem installierten Ort der API mit Suffix „/apidocs“, wie etwa <http://127.0.0.1:5002/apidocs>. Dort findet man die Kategorie events. Unter dieser sind zwei Funktionen gespeichert, zum ersten die „GET“ Funktion welche einem ein Beispiel API Event liefert. Zusätzlich auch den curl Befehl den man in der Konsole absetzen kann.

```
curl -X GET --header 'Accept: application/json'
'http://127.0.0.1:5002/oats/api/events'
```

Als Antwort darauf erhält man:

```
{
  "events": [
    {
      "data": 0,
      "event_name": "default_api_event",
      "host": "None",
      "severity": 1,
      "timestamp": 1527146905,
      "type": "api"
    }
  ]
}
```

Für den POST Befehl kann man das gleiche erhalten:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' -d '{ "data": 1, "event_name": "default_api_event%27",
"host": "None", "severity": 1, "timestamp": 1528360255, "type": "API" }'
'http://127.0.0.1:5002/oats/api/event'
```

Auf den curl Befehl erhält man den Event als Antwort zurück um zu signalisieren, dass er erfolgreich war.



4.4.7 Testing der Workflows

Nun ist OATS bereits funktionsfähig und wird bei allfälligen Fehlern die richtigen Workflows ausführen. Zum Zeitpunkt des Endes der Bachelorarbeit sind folgende Workflows umgesetzt (funktionieren nur im BA-Testlab out of the box):

- INTERFACE_CHANGED
- OSPF_NEIGHBOR_DOWN
- PORT_FLAPPING
- OUT_DISCARDS_EXCEEDED

Ein INTERFACE_CHANGED Event kann in der BA Testumgebung generiert werden, indem ein Subinterface auf einem der Router R11-R16 runtergefahren wird.:

```
R11#conf t
R11(config)#interface GigabitEthernet2.12
R11(config-subif)#shutdown
```

Dies sendet eine INTERFACE_CHANGED syslog-message zu napalm-logs und löst so den Troubleshooting-Prozess aus. Am Ende des Workflows wird das Interface automatisch wieder hochgefahren.

Analog kann der OSPF_NEIGHBOR_DOWN Workflow ausgelöst werden, indem der OSPF Prozess eines Routers runtergefahren wird:

```
R14#conf t
R14(config)#router ospf 1
R14(config-router)#shutdown
```

Sobald die OSPF dead timer der umliegenden Router ausgelaufen sind, lösen die dazugehörigen syslog-messages den OSPF-NEIGHBOR_DOWN Troubleshooting-Prozess aus. Am Ende des Workflows sollte der OSPF-Prozess wieder hochgefahren werden.

Zum Testen von PORT_FLAPPING kann ein Interface analog zu oben, repetitiv runter- und hochgefahren werden, um ein Port-Flapping zu simulieren.

Für OUT_DISCARDS muss lediglich traffic vom einen PC welcher an sw01 oder sw02 angeschlossen ist an den anderen generiert werden, welcher dafür sorgt, dass auf einem Interface Pakete gedroppt werden. Dafür werden folgende Befehle benötigt:

1. PC

```
student@ws2103c-23:~$ iperf3 -
```

2. PC

```
student@ws2103c-24:~$ iperf3 -c 192.168.2.10 -u -l 1460 -b 105M
```

Damit wird genügend traffic generiert, sodass Pakete gedroppt werden. OATS erkennt dies dank Streaming-Telemetry, Netflow und Kafka-Streams und lädt dann eine Konfiguration auf das Gerät, welches den traffic zwischen diesen zwei hosts mit dem Destination-Port der iperf3 Applikation, drosselt.



4.5 Development

Der gesamte Code den wir im Verlaufe dieser Bachelorarbeit geschrieben haben, wurde ausführlich dokumentiert, deswegen werden wir hier nicht den Code selbst dokumentieren sondern verweisen auf die Kommentare beim Code selbst.

Um OATS neue Troubleshooting-Workflows hinzuzufügen müssen folgende Komponenten erweitert/ neu entwickelt werden:

4.5.1 Syslog

Als erstes muss eine syslog-message identifiziert werden, mit der ein Troubleshooting-Prozess ausgelöst werden kann. Existiert eine solche message muss napalm-logs erweitert werden, damit die übermittelten Daten aus der message extrahiert werden können.

4.5.2 Napalm-Logs

Wie napalm-logs um neue syslog-messages erweitert werden kann ist hier beschrieben:

http://napalm-logs.readthedocs.io/en/latest/developers/device_profiles.html

Im Rahmen der SA wurden bereits mehrere solche Profile geschrieben. Die Profile, sowie auch der Prefix-Parser für Cisco IOS befinden sich im OATS Github-Repository unter `napalm/logs/ios/`. Wir empfehlen, dass man sich beim Development von neuen Profilen an den existierenden orientiert. Auch bereits in napalm-logs existierende Profile von anderen Betriebssystemen können sich als hilfreich erweisen. Diese findet man unter:

https://github.com/napalm-automation/napalm-logs/tree/master/napalm_logs/config

4.5.3 OATS napalm logs Client

Der OATS napalm logs Client (`/OATS/napalm/scripts/oats_napalm_logs_client.py`) erfüllt folgende Funktionen:

- Empfangen der Daten von napalm-logs
- Formen und Weiterleiten dieser Daten in Form von Kafka-Records welche an Topics mit dem Namen des Syslog-Fehlers (z.B. OSPF_NEIGHBOR_DOWN, INTERFACE_CHANGED) weitergeleitet werden.

4.5.4 Kafka Consumer

Für jede neue Art von Event (sei es Syslog, Streaming-Telemetry oder Netflow-basiert), welche dem System hinzugefügt wird, muss ein neuer Kafka-Consumer geschrieben. Beispiele dafür sind unter `/OATS/kafka/` abgelegt. Dabei muss jeweils aus dem Kafka-Topic mit dem richtigen Namen gelesen werden. Die Namen werden bei Streaming-Telemetry direkt aus der Konfigurationsdatei gelesen und bei syslog basieren diese auf dem Namen des Fehlers der syslog-message.

Innerhalb dieser Consumer hat man zwei Möglichkeiten: Erstellen eines Salt-Events oder das Korrelieren der Daten.

Für Netflow existiert noch kein Kafka-Consumer, weshalb das Naming dafür von den neuen Entwicklern festgelegt werden kann.



Konstruktion der Salt-Events

Syslog Events

Alle Syslog Salt-events, welche an den Salt-event-bus übermittelt werden, haben folgende Form:

`syslog/*/OSPF_NEIGHBOR_DOWN/dead_timer_expired`

Anhand dieses Event-Namens wird entschieden, welcher Troubleshooting-Workflow ausgeführt werden soll.

Um zu verstehen woher OSPF_NEIGHBOR_DOWN und dead_timer_expired extrahiert werden, muss man verstehen, wie die Daten von napalm-logs modelliert werden. Als Beispiel dient hier eine OSPF-NEIGHBOR_DOWN syslog-message:

```
{
  "error": "OSPF_NEIGHBOR_DOWN",
  "facility": 3,
  "host": "vmx01",
  "ip": "127.0.0.1",
  "message_details": {
    "date": "Jun 21",
    "facility": 3,
    "host": "vmx01",
    "hostPrefix": null,
    "message": "OSPF neighbor 1.1.1.1 (realm ospf-v2 ge-0/0/0.0 area 0.0.0.0) state changed from Full to Down due to InActiveTimer (event reason: BFD session timed out and neighbor was declared dead)",
    "pri": "29",
    "processId": "2902",
    "processName": "rpd",
    "severity": 5,
    "tag": "RPD_OSPF_NBRDOWN",
    "time": "14:03:12"
  },
  "os": "junos",
  "severity": 5,
  "timestamp": 1498053792,
  "yang_message": {
    "network-instances": {
      "network-instance": {
        "global": {
          "protocols": {
            "protocol": {
              "ospf": {
                "ospfv2": {
                  "areas": {
                    "area": {
                      "area": {
                        "interfaces": {
                          "interface": {
                            "GigabitEthernet2.12": {
                              "neighbors": {
                                "neighbor": {
                                  "172.16.12.1": {
                                    "state": {
                                      "adjacency-state": "DOWN",
                                      "adjacency-state-change-reason": "Dead timer expired",
                                      "adjacency-state-change-reason-message": "DEAD"
                                    }
                                  }
                                }
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  },
  "yang_model": "openconfig-ospf"
}
```

Die message ist als Yang-Modell modelliert und wird im Oatsclient serialisiert (bzw. in ein python-dict transformiert).

OSPF_NEIGHBOR_DOWN entspricht dem Feld `error` und `dead_timer_expired` dem Feld `adjacency-state-change-reason`. Diese werden im Kafka-Consumer jeweils extrahiert und dem Salt-Event-Namen angefügt.

Soll das Event direkt an den Salt-master weitergeleitet werden, so muss

`EventProcessor.process_event(...)` unter `/OATS/kafka/oats_kafka_helpers` mit den geforderten Parametern ausgeführt werden. Standardmässig werden alle Events die so übermittelt werden an



den Salt-master geschickt und in die InfluxDB geschrieben. Soll dies nicht geschehen, steht dafür je ein optionaler Parameter bereit mit welchem dieses Verhalten geändert werden kann.

Streaming-Telemetry Events

Bei den Streaming-Telemetry Daten wird direkt im OATS-Konfigurationsfile unter `/etc/oats/config.yaml` angegeben in welches Kafka-Topic die Events geschrieben werden und wie diese Events heißen sollen. Wir empfehlen das Verwenden von möglichst eindeutigen Topic-Namen. Das Event-Naming folgt am besten dem Schema `streaming-telemetry/*/<event-string>`. Beim ersten umgesetzten Workflow war der Event-Name: `streaming-telemetry/*/out-discard-event`.

Aggregieren von Events

Wie bereits erwähnt ist es auch möglich Events mit vordefinierten Funktionen zu aggregieren. Dafür stehen unter `/OATS/kafka/oats_kafka_helpers` 3 entwickelte Funktionen zur Verfügung:

- **aggregate_identical:** zum Aggregieren gleichartiger Events
- **aggregate_distinct:** zum Aggregieren unterschiedlicher Events.
- **compress:** zum Komprimieren von Events.

Diese Funktionen müssen als Threads gestartet werden, damit sie in der Lage sind die Anzahl der Events zu zählen. Beispiele dafür sind wieder unter `/OATS/kafka/` einsehbar.

Am Ende der Korrelation wird jeweils evaluiert, ob die Korrelation erfolgreich war. Im Erfolgsfall wird ein anderes Event übermittelt als bei einem Fehlversuch.

Hier wäre es denkbar weitere Korrelations-Funktionen zu entwickeln, welche in der Domainanalyse beschrieben werden.



4.5.5 Salt

Salt erfüllt innerhalb von OATS folgende Funktionen:

- Entgegennehmen von Events
- Ausführen von Workflows

Hier wird nur erklärt, welche Dateien man erweitern/ändern/neu erstellen muss um OATS zu erweitern. Für Informationen wie Salt funktioniert verweisen wir auf:

<https://docs.saltstack.com/en/latest/>

Um Salt für OATS zu erweitern sind folgende Schritte nötig:

Salt Master File

Die Salt-master Datei muss um das neue Event erweitert werden, auf welches gehorcht werden soll. Es ist wichtig, dass das im jeweiligen Kafka-Consumer erstellte und das in der master Datei angegebene Event übereinstimmen.

Die master Datei befindet sich standardmässig unter `/etc/salt`.

Die Events werden direct im Salt-reactor innerhalb der Salt-master Datei wie folgt angegeben:

reactor:

- 'syslog/*/INTERFACE_CHANGED/down':
- /srv/saltstack/reactor/ifdown_workflow.sls

Das Event auf das gehorcht werden soll:

`syslog/*/INTERFACE_CHANGED/down`

Der Salt-state der als Reaktion auf das Event ausgeführt werden soll:

`/srv/saltstack/reactor/ifdown_workflow.sls`

Salt Reactor

Im Salt-reactor directory werden state files abgelegt, welche Salt Funktionen (bzw. die in OATS geschriebenen Workflows) ausführen. Sie befinden sich wie man oben bereits sieht unter `/srv/saltstack/reactor`.

Ein state file welches einen INTERFACE_DOWN-Workflow ausführt sieht folgendermassen aus:

```
{% set event_data = data['data'] %}

ifdown_workflow:
  runner.tshoot.ifdown:
    - host: {{ event_data['host'] }}
    - yang_message: {{ event_data['data']['yang_message'] }}
    - error: {{ event_data['data']['error'] }}
    - tag: {{ event_data['data']['message_details']['tag'] }}
    - current_case: {{ event_data['case'] }}
```

Die in geschweiften Klammern vorkommenden Ausdrücke sind Jinja-statements. Mit Jinja werden dem jeweiligen Event die Daten entnommen und dem Workflow übergeben. Mehr Informationen zu Jinja findet man unter:

<http://jinja.pocoo.org/docs/2.10/>

Der Name (ifdown_workflow) ist frei wählbar.

`runner.tshoot.ifdown` gibt an welche Funktion in welchem file in welchem directory ausgeführt werden soll:

`runner`: gibt an, dass es sich beim auszuführenden file um einen Salt-runner handelt

(<https://docs.saltstack.com/en/latest/ref/runners/>) und sich im directory

`/srv/saltstack/salt/_runners` befindet.



`tshoot`: gibt an wie das python-file heisst (`tshoot.py`).

`ifdown`: gibt an welche Funktion innerhalb des files ausgeführt werden soll.

Zuunterst folgt eine Liste von Parametern, die von der `ifdown`-Funktion übernommen werden, d.h. Die Funktion `ifdown` im file `tshoot.py` hat folgende Form:

```
ifdown(host, origin_ip, yang_message, error, tag, current_case=None)
```

Salt Runners / Workflows

Die Workflows werden in OATS durch Salt-runners abgebildet, d.h. die gesamte Logik passiert innerhalb von python-files im `_runners` directory.

Um einen neuen Workflow zu schreiben muss lediglich eine neue Funktion innerhalb von `tshoot.py` erstellt werden, welche von einem reactor state file ausgeführt wird.

Es wird empfohlen die bereits bestehenden workflows `ifdown` und `ospf_neighbor_down` als Template zu nehmen. Auch existieren bereits eine Reihe von Helfer-Methoden innerhalb von `oatssalthelpers.py`. Dort werden mittels `salt.cmd` und `salt.execute` Salt runner und execution modules ausgeführt, mit welchen z.B. mittels `napalm` Informationen von Geräten gesammelt werden können. Siehe auch:

https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.napalm_network.html

Überlegungen für die Zukunft

Das Ausführen von bereits existierenden Workflows ist durch Salt sehr einfach gestaltet. Das Schreiben von neuen Workflows kann allerdings beliebig komplex werden, weswegen Programmierwissen vorausgesetzt werden muss.

Der Aufwand innerhalb eines Workflows kann massiv verringert werden, wenn ein Teil der Datensammlung bereits im Kafka-Consumer (durch Korrelation) erledigt wird. So wird im `ospf_nbr_down` Workflow lediglich der OSPF-Prozess auf dem betroffenen Router neu gestartet und überprüft ob dies das Problem behoben hat.

Wenn ein Grossteil der Daten bereits ausserhalb von Salt gesammelt wird, wäre es vorstellbar, dass man auch hier die Workflows direkt als YAML-files ablegen könnte. Dafür müsste man z.B. schon wissen auf welchem Gerät welches Problem besteht, bevor ein Event an Salt übermittelt wird. Ein ungelöstes Problem von Salt, ist die Geschwindigkeit der Netzwerkzugriffe. Für die in der SA entwickelten Workflows reichte die Geschwindigkeit. Für den `OUT_DISCARDS` Workflow stösst Salt hier an seine Grenzen, da das Laden der neuen Konfiguration ca. 30-45 Sekunden dauert. Bis dies geschehen ist, könnte das Problem schon wieder weg sein und der Workflow hat keine Wirkung. Für die Zukunft müsste hier evaluiert werden, ob es eventuell ein System gibt, welches schneller auf Events reagieren kann.

4.5.6 Streaming-Telemetry

Für Streaming-Telemetry wurde ein Client geschrieben, welcher in der Lage ist aufgrund einer Konfigurationsdatei Verbindungen zu verschiedenen Hosts aufzubauen und regelmässig Daten von diesen zu empfangen.

Es wurde versucht diese Konfigurationsdatei so zu gestalten, dass der Code nicht angepasst werden muss. Deshalb verweisen wir hier auf das Kapitel 4.3 (Konfigurationsanleitung).

Sollte der Code doch noch angepasst werden, so findet man den Code unter `/OATS/telemetry/`.



4.5.7 Systemd

Sollten neue Skripts hinzugefügt werden, welche auch als Daemons zusammen mit den anderen Skripten im Hintergrund auf dem Server ausgeführt werden, so müssen diese mittels systemd-files daemonisiert werden.

Beispiele dafür befinden sich unter `/OATS/systemd/` und sind einfach zu schreiben. Es darf nur nicht vergessen werden, den neuen Service in `oats.target` hinzuzufügen, damit das neue Skript zusammen mit allen anderen gestartet wird.

4.5.8 Kafka-Streams

Der Kafka-Streams Client in OATS wurde im Gegensatz zum Rest nicht mit Python sondern mit Java entwickelt.

Ein sehr gutes Tutorial dazu findet man unter <https://kafka.apache.org/documentation/streams/> und unser Code befindet sich unter `/OATS/streaming-clients/oats-generic-kafka-streams/`.

4.5.9 Netbox

Falls man aus Netbox weitere Informationen extrahieren möchte, als die bereits implementierten Funktionen aus `oatsnb` liefern, müssen man diese neu programmieren. Da hier mit dem `pynetbox` Package gearbeitet wird, verweisen wir auf dessen Dokumentation, welche unter <http://pynetbox.readthedocs.io/en/latest/> zu finden ist.



III. Glossar

1. Begriffe

Aktor	Ein Element, das eine Eingangsgröße in eine andersartige Ausgangsgröße umwandelt, um einen gewünschten Effekt hervorzurufen.
Construction	Dritte Phase von RUP. Dient der Entwicklung und dem Testing des Systems.
Daemon	Programm, welches im Hintergrund abläuft und gewisse Dienste zur Verfügung stellt.
Elaboration	Zweite Phase von RUP. Dient zum Erstellen eines Architekturprototyps, einer Domainanalyse und dem Aufsetzen der Anforderungen an das System. Zusätzlich wird in dieser Zeit die nachfolgende Construction Phase geplant (Anm: Die Bachelorarbeit wurde nicht vollständig mit RUP durchgeführt, sondern enthält nur einige Elemente davon)
Event	Dt. Ereignis. Ein spezielles Ereignis auf Netzwerkgeräten (z.B. unbeabsichtigtes Runterfahren des Geräts), welches einen Troubleshooting-Prozess auslöst.
Event Driven Automation	Beschreibt das automatische Ausführen von Tasks als Reaktion auf gewisse Events in einem System. So können z.B. manuelle Änderungen im System erkannt werden und rückgängig gemacht werden.
Host	Dienstrechner der Teil eines Netzwerks ist
Inception	Erste Phase von RUP. Dient zur Findung eines Konzepts (Beschreibung der Funktionalität des zu entwickelnden Systems) und der Zielsetzung.
Interface (Netzwerk)	Dt. Netzwerkkarte. Dient der Verbindung zwischen 2 Geräten. Verbindet im Netzwerkumfeld vor allem Switches und Router.
Interface (Software)	Dt. Schnittstelle. Bezeichnet Schnittstellen zwischen zwei oder mehreren Systemen.
Network-Health	Aktueller Zustand des Netzwerks in dem OATS eingesetzt wird. Der Zustand leitet sich aus der offenen Anzahl Problemfälle im Netz ab.
Open Source	Bezeichnet Software deren Quellcode öffentlich verfügbar ist und gratis verwendet werden kann.
Problemfall	Auftreten eines Problems im Netzwerk in dem OATS eingesetzt wird.
Salt-engine	Extern laufende Salt-Prozesse, z.B. der Salt-reactor oder der Salt-Event-Bus
Salt-Event	Die Repräsentation eines Events welches von Syslog ausgelöst und von napalm-logs verarbeitet wurde, innerhalb von Salt
Salt-Event-Bus	Die ZeroMQ Message-Queue für Salt-Events.
Salt-master	Läuft in OATS 1x auf dem zentralen Server. Zuständig für die Orchestrierung und Kommunikation mit den Salt-minions und proxies.
Salt-minion	Objekte die eine verkleinerte Version von Salt ausführen um die lokale Ausführung von Befehlen und Kommunikation mit dem master zu ermöglichen.



Salt-proxy	Sind dem Salt-master untergeordnet und dienen als Proxy (dt. Vermittler) zu je einem Netzwerkgerät. Sie verfügen dabei über die nötigen Informationen über das Netzwerkgerät, um mit diesem zu kommunizieren.
Salt-reactor	Listener für Salt-Events auf dem Salt-Event-Bus. Führt als Reaktion auf solche Events einen Workflow aus
Sensor	Detektor von Events im Netzwerk.
Skript	Beschreibt während der Arbeit entwickelte Python-Programme.
Streaming-Daten	Daten, welche kontinuierlich, in regelmässigen Abständen an ein System gesendet werden. Beschränkt sich nicht nur auf Streaming-Telemetry.
Streaming-Telemetry	Ein moderner Ansatz um SNMP-basiertes Monitoring zu ersetzen. Dabei liefert ein Netzwerkgerät kontinuierlich Daten, welche an einer zentralen Stelle gesammelt und analysiert werden können.
Subscription	dt. Abonnement
Techniker	Mitarbeiter in einer Firma in dem OATS eingesetzt wird. Verfügt über vertieftes Wissen über Netzwerke, aber nicht über Programmierkenntnisse.
Tool	Dt. Dienstprogramm. Ein Programm welches systemnahe Aufgaben ausführt. Bietet eine API mit der verschiedene Funktionen ausgeführt werden können. So werden in OATS z.B. Zugriffe auf die Netzwerkgeräte durch napalm ausgeführt, wobei der Anwender sich nicht darum kümmern muss, um welches Gerät es sich handelt.
Traffic	Netzwerkverkehr
Transition	Vierte und Letzte Phase von RUP. Dient als Übergabephase in welcher das Projekt abgeschlossen wird. Dabei liegt das Hauptaugenmerk auf der Dokumentation und dem Deployment der entwickelten Software.
Troubleshooting	Problemlösungsstrategie. Umfasst das logische und systematische Suchen nach der Quelle eines Problems. Wird im Netzwerkumfeld vor allem im Zusammenhang mit Netzwerkgeräten (Router Switches) benutzt.
Use Case	Ein Anwendungsfall, welcher mögliche Szenarien beinhaltet, welche ein Akteur über das System erfolgreich durchführen möchte.
Workflow	Dt. Arbeitsablauf. Beschreibt eine Reihenfolge von Arbeitsvorgängen.



2. Abkürzungen

ACL	Access Control List
API	Application Programming Interface
ARP	Address Resolution Protocol
BDPU	Bridge Protocol Data Units
BGP	Border Gateway Protocol
CDP	Cisco Discovery Protocol
CIR	Comitted Information Rate
CLI	Command Line Interface
CRC	Cyclic Redundancy Check
DHCP	Dynamic Host Configuration Protocol
EIGRP	Enhanced Interior Gateway Protocol
HSRP	Hot Standby Router Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task force
IF	Interface
INS	Institute for Networked Solutions (HSR)
IP SLA	Internet Protocol Service Level Agreement
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
JSON	Javascript Object Notation
LACP	Link Aggregation Protocol
MAC	Media Access Controll (Address)
MIB	Management Information Base
MTU	Maximum Transmission Unit
NAPALM	Network Automation and Programmability Abstraction Layer with Multivendor support
NAT	Network Address Translation
NETCONF	Network Configuration Protocol
NFR	Non Functional Requirements
NOSQL	Not Only SQL
nv	Nico Vinzens
OATS	Open Source Automated Troubleshooting System
OS	Operating System (dt. Betriebssystem)
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PAgP	Port Aggregation Protocol
PAT	Port Address Translation
PDU	Packet Data Unit
pip	pip installs packages
QoS	Quality of Service
rj	Raphael Jöhl
RPC	Remote Procedure Call
RUP	Rational Unified Process
SA	Studienarbeit
SLAAC	Stateless Address Autoconfiguration
SLACK	Searchable Log of All Conversation and Knowledge



SNMP	Simple Network Managing Protocol
SQL	Structured Query Language
SSH	Secure Shell
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
UC	Use Case
UDP	User Datagram Protocol
UID	Unique Identifier
VLAN	Virtual Local Area Network
VRF	Virtual Routing and Forwarding
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language (ursprünglich: Yet Another Markup Language)
YAQL	Yet Another Query Language



IV. Bibliographie

1. Quellenverzeichnis

- [1] wikibooks, «Linux-Praxisbuch: Syslog,» 04 10 2017. [Online]. Available: https://de.wikibooks.org/wiki/Linux-Praxisbuch:_Syslog.
- [2] OpenConfig, «Vendor-neutral, model-driven network management designed by users,» 15 10 2017. [Online]. Available: <http://openconfig.net/>.
- [3] Cisco, «Cisco Netflow RFC,» Cisco, 2018. [Online]. Available: <https://www.ietf.org/rfc/rfc3954.txt>. [Zugriff am 6 6 2018].
- [4] D. Barroso, «YANG for Dummies,» 04 10 2017. [Online]. Available: <https://napalm-automation.net/yang-for-dummies/>.
- [5] A. Müller, «Event Correlation Engine,» ETH Zürich, Zürich, 2009.
- [6] influxdata, «influxdata - The modern engine for Metrics and Events,» influxdata, 07 03 2018. [Online]. Available: <https://www.influxdata.com/>. [Zugriff am 07 03 2018].
- [7] Apache, «Apache Kafka,» Apache, 26 03 2018. [Online]. Available: <https://kafka.apache.org/>. [Zugriff am 26 03 2018].
- [8] M. Ulinic, «Using napalm-logs for event-driven network automation and orchestration,» 15 10 2017. [Online]. Available: <https://napalm-automation.net/napalm-logs-released/>.
- [9] Wikipedia, «ZeroMQ,» 15 10 2017. [Online]. Available: <https://en.wikipedia.org/wiki/ZeroMQ>.
- [10] VerizonDigital, «github.com/VerizonDigital/vflow,» VerizonDigital, 11 01 18. [Online]. Available: <https://github.com/VerizonDigital/vflow>. [Zugriff am 06 06 2018].
- [11] Saltstack, «Salt Open Source,» 15 10 2017. [Online]. Available: <https://saltstack.com/salt-open-source/>.
- [12] D. Barroso, «Welcome to NAPALM's documentation!,» NAPALM, 15 03 2018. [Online]. Available: <https://napalm.readthedocs.io/en/latest/>. [Zugriff am 15 03 2018].
- [13] Grafana, «Grafana Labs,» Grafana, 11 06 2018. [Online]. Available: <https://grafana.com/>. [Zugriff am 11 06 2018].
- [14] Netbox, «Netbox - Introduction,» Netbox, 07 03 2018. [Online]. Available: <https://netbox.readthedocs.io/en/latest/>. [Zugriff am 07 03 2018].
- [15] PostgreSQL, «PostgreSQL,» 14 10 2017. [Online]. Available: <https://www.postgresql.org/about/>.
- [16] IETF, «RFC 792 - Internet Control Message Protocol,» 03 10 2017. [Online]. Available: <https://tools.ietf.org/html/rfc792>.
- [17] Microsoft, «Tracert,» 06 10 2017. [Online]. Available: <https://technet.microsoft.com/en-us/library/cc940128.aspx>.
- [18] Linux/Unix, «Linux man-pages - traceroute,» 06 10 2017. [Online]. Available: <http://man7.org/linux/man-pages/man8/traceroute.8.html>.
- [19] IETF, «RFC 3411,» 03 10 2017. [Online]. Available: <https://tools.ietf.org/html/rfc3411>.
- [20] Cisco, «Building Scalable syslog Management Solutions,» 03 10 2017. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/services/high-availability/white_paper_c11-557812.html#wp9000392.
- [21] IETF, «RFC 6241 - The NETCONF Protocol,» 03 10 2017. [Online]. Available: <https://tools.ietf.org/html/rfc6241>.



- [22] IETF, «RFC 5424 - The Syslog Protocol,» 01 10 2017. [Online]. Available:
] <https://tools.ietf.org/html/rfc5424>.
- [23] IETF, «RFC 5424 - The Syslog Protocol,» 04 10 2017. [Online]. Available:
] <https://tools.ietf.org/html/rfc5424>.
- [24] InterProjektWiki, «InterprojektWiki,» 07 10 2017. [Online]. Available: <https://pliki.ip-sa.pl/wiki/Wiki.jsp?page=NetFlow>.
- [25] IETF, «RFC 3954 - Cisco Systems NetFlow Services Export Version 9,» 07 10 2017. [Online].
] Available: <https://tools.ietf.org/html/rfc3954>.
- [26] B. Mitchell, «What Is a Network Sniffer?,» 10 05 2017. [Online]. Available:
] <https://www.lifewire.com/definition-of-sniffer-817996>.
- [27] Cisco, «IP SLA Fundamentals,» 09 10 2017. [Online]. Available:
] <https://learningnetwork.cisco.com/blogs/vip-perspectives/2017/06/13/ip-sla-fundamentals>.
- [28] Cisco Canada, «SlideShare,» Cisco Canada, 11 03 2018. [Online]. Available:
] <https://de.slideshare.net/CiscoCanada/model-driven-telemetry>. [Zugriff am 11 03 2018].
- [29] GRPC, «grpc - A high performance, open-source universal RPC Framework,» GRPC, 07 03 2018.
] [Online]. Available: <https://grpc.io/>. [Zugriff am 07 03 2018].
- [30] Google, «Google Developers,» Google, 11 03 2018. [Online]. Available:
] <https://developers.google.com/protocol-buffers/>. [Zugriff am 11 03 2018].
- [31] Cisco, «Model-Driven Telemetry and Analytics,» 04 03 2018. [Online]. Available:
] <https://www.ciscolive.com/global/on-demand-library/?search=streaming%20telemetry&search.event=ciscoliveemea2018#/session/1501752087866001Uyok>.
- [32] Cisco, «github,» Cisco, 11 03 2018. [Online]. Available:
] <https://github.com/YangModels/yang/tree/master/vendor/cisco>. [Zugriff am 11 03 2018].
- [33] S. Cadora, «@XRDOCS,» Cisco, 21 03 2018. [Online]. Available:
] <https://xrdocs.github.io/telemetry/tutorials/2016-07-21-configuring-model-driven-telemetry-mdt/>. [Zugriff am 21 03 2018].
- [34] Microsoft, «Microsoft Support,» 09 10 2017. [Online]. Available:
] <https://support.microsoft.com/en-us/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained>.
- [35] E. Rivard, «Pearson IT Certification,» 09 10 2017. [Online]. Available:
] <http://www.pearsonitcertification.com/articles/article.aspx?p=1730891>.
- [36] Cisco, «Troubleshooting OSPF,» 09 10 2017. [Online]. Available:
] <https://www.cisco.com/c/dam/en/us/support/docs/ip/open-shortest-path-first-ospf/12151-trouble-main-00.jpeg>.
- [37] OpenTSDB, «OpenTSDB - The Scalable Time Series Database,» OpenTSDB, 07 03 2018. [Online].
] Available: <http://opentsdb.net/>. [Zugriff am 07 03 2018].
- [38] neo4j, «neo4j home,» neo4j, 14 3 2018. [Online]. Available: <https://neo4j.com/>. [Zugriff am 14 3 2018].
- [39] Cisco, «APIC-EM about,» Cisco, 2018. [Online]. Available:
] https://www.cisco.com/c/en/us/td/docs/cloud-systems-management/application-policy-infrastructure-controller-enterprise-module/1-0-x/deploy-guide/b_apic-em_deploy_guide_v_1_0_0_x/b_apic-em_deploy_guide_v_1_0_0_x_chapter_01.html. [Zugriff am 12 03 2018].
- [40] Saltstack, «Saltstack Doc,» 16 10 2017. [Online]. Available:
] https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.slack_notify.html.
- [41] ISO, «ISO/IEC 9126,» 30 09 2017. [Online]. Available: <https://www.iso.org/standard/22749.html>.



-]
- [42 D. Barroso, «Welcome to the NAPALM Documentation,» 15 10 2017. [Online]. Available:
] <http://napalm.readthedocs.io/en/latest/>.
- [43 N. Litzel, «BigData Insider,» BigData-Insider, 01 09 2016. [Online]. Available:
] <https://www.bigdata-insider.de/was-ist-hadoop-a-587448/>. [Zugriff am 24 03 2018].
- [44 Apache, «Wikipedia - Apache Hadoop,» Apache Hadoop, 26 03 2018. [Online]. Available:
] https://en.wikipedia.org/wiki/Apache_Hadoop. [Zugriff am 26 03 2018].
- [45 Apache, «Apache - Hadoop,» Apache, 24 03 2018. [Online]. Available:
] <http://hadoop.apache.org/>. [Zugriff am 24 03 2018].
- [46 Splunk, «Splunk,» Splunk, 24 03 2018. [Online]. Available: https://www.splunk.com/de_de.
] [Zugriff am 24 03 2018].
- [47 pandas, «pandas - Python Data Analysis Library,» pandas, 24 03 2018. [Online]. Available:
] <https://pandas.pydata.org/>. [Zugriff am 24 03 2018].
- [48 Confluent, «Confluent - Kafka Streams API,» Confluent, 26 03 2018. [Online]. Available:
] <https://docs.confluent.io/current/streams/introduction.html#the-kafka-streams-api-in-a-nutshell>. [Zugriff am 26 03 2018].
- [49 Winton, «github,» Winton-Kafka-Streams, 26 03 2018. [Online]. Available:
] <https://github.com/wintoncode/winton-kafka-streams>. [Zugriff am 26 03 2018].
- [50 A. Shaw, «Medium,» 14 10 2017. [Online]. Available:
] <https://medium.com/@anthonyjpshaw/ansible-v-s-salt-saltstack-v-s-stackstorm-3d8f57149368>.
- [51 Red Hat Ansible, «ansible,» 14 10 2017. [Online]. Available: <https://www.ansible.com/>.
]
- [52 Stackstorm, «Stackstorm,» 14 10 2017. [Online]. Available: <https://stackstorm.com/>.
]
- [53 Slack, «Slack,» 16 10 2017. [Online]. Available: <https://slack.com/>.
]
- [54 MongoDB, «MongoDB,» 14 10 2017. [Online]. Available: <https://www.mongodb.com>.
]
- [55 Cisco Learning Network, «The 20 most common issues in Routing,» 30 09 2017. [Online].
] Available: <https://learningnetwork.cisco.com/thread/63944>.
- [56 VMS, «VMS - find a new path,» 01 10 2017. [Online]. Available:
] <https://www.vms4x4.com/products/60-minute-labour-charge>.
- [58 Cisco Tail-f, «Tail-f-Systems,» 03 10 2017. [Online]. Available: <http://www.tail-f.com/wordpress/wp-content/uploads/2013/03/Tail-f-NETCONF-YANG-Service-Automation-LISA-Usenix-2011.pdf>.



2. Abbildungsverzeichnis

Abbildung 1: Streaming-Telemetry Yang-Modell	9
Abbildung 2: virtuelles Lab-Netz	11
Abbildung 3: physikalisches Lab-Netz	11
Abbildung 4: OATS-Architektur	12
Abbildung 5: minimale OATS-Konfigurationsdatei.....	13
Abbildung 6: OATS-Grafana.....	15
Abbildung 7: 3: ping -n5 auf hsr.ch	21
Abbildung 8: tracert auf google.com.....	24
Abbildung 9: SNMP-Paket Aufbau [9]	25
Abbildung 10: Trap-PDU-Header	26
Abbildung 11: YANG Modelldefinition	29
Abbildung 12: Beispiel eines YANG Datensatzes.....	30
Abbildung 13: Importieren und Erweitern eines bestehenden Modells.....	30
Abbildung 14: NetFlow Funktionsweise [9]	34
Abbildung 15: Telemetry-Modell	37
Abbildung 16: Interface Yang-Model	38
Abbildung 17: IOX-XR Telemetry-Konfiguration.....	39
Abbildung 18: traditionelle Netzwerküberwachung	40
Abbildung 19: Netzwerküberwachung mittels Streaming-Telemetry.....	40
Abbildung 20: OSPF Troubleshooting-Prozess	46
Abbildung 21: EIGRP Troubleshooting Prozess	47
Abbildung 22: BGP Troubleshooting-Prozess.....	49
Abbildung 23: Netbox-UI.....	58
Abbildung 24: APIC-EM Features	59
Abbildung 25: Campus-Netz	73
Abbildung 26: Campus-Netz mit BGP und VLAN	74
Abbildung 27: Architekturübersicht	75
Abbildung 28: Use Case Diagramm	78
Abbildung 29: Use Case Diagramm nach Kategorien.....	79
Abbildung 30: Auszug der NAPALM Supportmatrix	92
Abbildung 31: get_interfaces call.....	93
Abbildung 32: Syslog-Meldung eines Cisco-Devices	94
Abbildung 33: Resultierende Meldung.....	94
Abbildung 34: Syslog Meldung eines Juniper-Devices	94
Abbildung 35: Apache Kafka.....	95
Abbildung 36: Kafka API	95
Abbildung 37: Kafka Topic-Partitionen.....	96
Abbildung 38: Kafka Streams Java-Applikation	99
Abbildung 39: Ansible.....	101
Abbildung 40: Ansible Tower	102
Abbildung 41: Ansible Support.....	102
Abbildung 42: Stackstorm	104
Abbildung 43: Stackstorm Infrakstruktur	105
Abbildung 44: Saltstack	107
Abbildung 45: Salt Funktionsweise	107
Abbildung 46: Salt Architektur	108
Abbildung 47: ZeroMQ Code Beispiel	111
Abbildung 48: Daten, welche in Netbox gespeichert werden können.....	112
Abbildung 49: Grafana Dashboard	113
Abbildung 50: Architektur: Studienarbeit	114



Abbildung 51: Architektur Bachelorarbeit	116
Abbildung 52: Architekturdiagramm.....	121
Abbildung 53: Deployment Diagramm.....	128
Abbildung 54: Netflow-Daten in der InfluxDB.....	131
Abbildung 55: Flow-Monitore auf den Interfaces	149
Abbildung 56: Flow Exporter & Monitore	149
Abbildung 57: Flow-Records.....	149



V. Anhänge



Projekt: Automatisiertes Troubleshooting

Raphael Jöhl
Nico Vinzens



Anhang A: Projektplan

Änderungsgeschichte

Datum	Version	Änderung	Autor
21.02.2018	1.0	Erstellen des Dokuments	nv
26.02.2018	1.1	Erweiterung gemäss erstem Meeting	rj
27.02.2018	1.2	Ergänzungen	nv, rj
28.02.2018	1.3	Ergänzungen	nv, rj
01.03.2018	1.4	Ergänzungen	nv, rj
06.04.2018	1.6	Anpassung Aufgabenstellung	nv
29.05.2018	2.0	Überarbeitung vor Abgabe	nv



A.1 Einführung

A.1.1 Zweck

Während der Bachelorarbeit vom Frühlingsemester 2018 soll das während dem vorigen Herbstsemester entwickelte OATS (Open Source Automated Troubleshooting) erweitert werden. In diesem Dokument wurde zu diesem Zweck eine Planung, sowie Arbeitspakete erstellt, um eine erste Vorstellung darüber zu haben, wie das Projekt ablaufen soll.

A.1.2 Gültigkeitsbereich

Das Dokument soll nur HSR intern verwendet werden.

A.2 Projektbeschreibung

A.2.1 Zweck und Ziel

A.2.2 Aufgabenstellung SA

In der Studienarbeit „Automated Troubleshooting“ wurde folgende Aufgabenstellung bearbeitet:

Viele Arbeitsschritte bei Fehlerbehebungen könnten eigentlich automatisiert werden. So arbeitet eine nette Stimme bei der Support Hotline aller grossen Telekom Konzerne zuerst die Checkliste ab. Das gleiche hat auch in den anderen IT Bereichen seine Gültigkeit. So sammelt ein Netzwerk Engineer bei einem Support Ticket zuerst meistens nur Informationen. Wo befindet sich das Gerät? In welchem Subnetz ist es? Kann ich es via ICMP erreichen? Solche Arbeiten eignen sich sehr gut zum Automatisieren. So können bei einem Fehlerfall die üblichen Tasks bereits abgearbeitet werden. Sind genügend Informationen vorhanden können automatisch Lösungsversuche gestartet werden oder das Problem temporär umgangen werden, so dass der Techniker am nächsten Tag ausgeschlafen die Bereinigung abschliessen kann.

Aufgabe: Entwickeln eines Systems, welches dank dem Konzept von «Event Driven Automation» das Troubleshooting automatisieren kann. Über Sensoren sollen Workflows angestossen werden, welche wiederum sogenannte Aktoren ansteuern. Es muss nicht eine komplette Eigenentwicklung geben, da es viele Komponenten bereits als Open Source Lösungen gibt. Im gesunden Rahmen sollen diese Projekte durch Integrationen und Anpassungen zu einer Gesamtlösung verschmelzen.

A.2.3 Aufgabenstellung BA

In einem Data-Center oder Campus-Netz passieren Fehler, welche oft erst spät erkannt werden, so dass sie negative Auswirkungen auf das Business haben, wie zB. dass sich ein Kunde beschwert, oder dass Geschäftsapplikationen nicht mehr funktionieren. Das Ziel der vorausgehenden Studienarbeit war, solche Störungen schnellstmöglich zu entdecken und automatisch zu beheben. Daraus ist OATS (**O**pen **S**ource **A**utomated **T**roubleshooting **S**ystem) entstanden, ein verteiltes System mit Sensoren, die Events auslösen, welche ihrerseits Troubleshooting Workflows auslösen, die versuchen das Problem automatisch zu beheben oder einen Techniker zu alarmieren.

In der nachfolgenden Bachelorarbeit soll OATS erweitert werden, damit eine grössere Bandbreite von Fehlern erkannt und automatisch behoben werden kann.



Konkret sollen:

- weitere Sensoren analysiert und ins System eingebunden werden
- die Events verschiedener Sensoren sollen intelligent korreliert werden können
- Troubleshooting Workflows sollen so erweitert werden, dass sie die Events der neuen Sensoren mitberücksichtigen
- Ein grafisches User-Interface soll den Zustand des Systems anzeigen

Um die Funktionstauglichkeit des erweiterten OATS Systems zu prüfen, sollen entsprechende Troubleshooting Use-Cases im Labornetzwerk ausgelöst und von OATS behandelt werden.

A.2.4 Lieferumfang

In der Studienarbeit die als Vorarbeit zu dieser Bachelorarbeit diente, wurde ein System erstellt, welches verschiedene Open-Source Lösungen sinnvoll verbindet, damit ein automatisches Troubleshooting möglich ist.

In dieser Arbeit wird das existierende System um weitere Funktionalitäten und Umfang erweitert, sowie bestehende Funktionalitäten überarbeitet.

A.2.5 Annahmen und Einschränkungen

Die Projektdauer ist auf 17 Wochen nach Beginn des Frühlingsemesters 2018 festgesetzt. Das Projekt sollte im Rahmen des geforderten Zeitaufwands von 720 Arbeitsstunden, bei 2 Projektmitgliedern realisierbar sein.

A.3 Projektorganisation

Studiengang:	Informatik
Semester:	FS 2018
Fachrichtung:	Application Design Cloud Solutions
Studenten:	Raphael Jöhl, Nico Vinzens
Verantwortlicher:	Beat Stettler
Betreuer:	Urs Baumann



A.4 Management Abläufe

A.4.1 Kostenvoranschlag

Das Projekt läuft während 17 Wochen. Insgesamt stehen für die Bachelorarbeit, mit zwei Studenten, 720 Stunden zur Verfügung. In den ersten 15 Wochen sollen dabei 20 Stunden pro Woche aufgewendet werden. In den letzten 2 Wochen, in denen keine Vorlesungen mehr stattfinden sollen je 40 Stunden aufgewendet werden. Dies führt ca. auf den insgesamt geforderten Arbeitsaufwand.

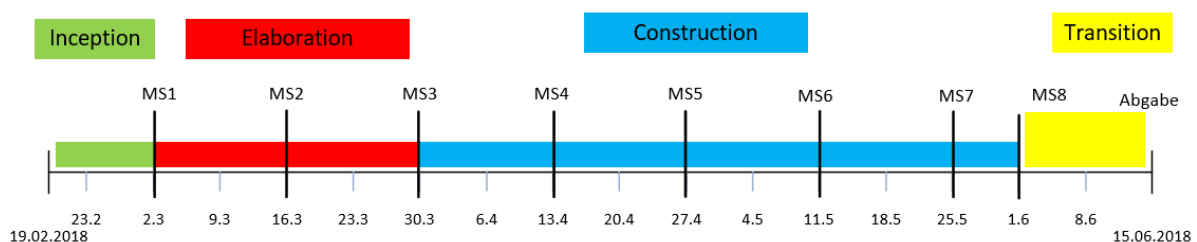
Projektdauer	17 Wochen
Projektmitglieder	2 Personen
Arbeitsstunden pro Person	15*20 + 2*40
Arbeitsstunden gesamt	760
Projektstart	19. Februar 2018
Projektende	15. Juni 2018

A.4.2 Arbeitsaufteilung

Raphael Jöhl: Datenbanken & Deployment

Nico Vinzens: Troubleshooting-Workflows & Datenverarbeitung

A.5 Zeitliche Planung



Jeweils am Donnerstag setzt sich das Team zusammen, um den Fortschritt der letzten Woche zu besprechen und um sicherzustellen ob alles was am nächsten Tag gezeigt werden soll, bereit ist.

A.5.1 Inception

Die Inception Phase dauert zwei Wochen und dient dem Erstellen eines Projektplans, Ausarbeiten von Arbeitspaketen und der Recherche von möglichen Erweiterungen von OATS.

A.5.2 Elaboration

Die Elaboration Phase dauert 4 Wochen. Am Ende der Elaboration sollen alle offenen Fragen geklärt sein. D.h. die Anforderungen sind bekannt, alle Tools sind bekannt und es existiert ein funktionierender Prototyp.

A.5.3 Construction

Die Construction Phase dauert 9 Wochen. Während dieser Zeit wird der Prototyp weiterentwickelt. Am Ende der Construction soll das System lauffähig sein und den Anforderungen entsprechen.

A.5.4 Transition

Die Transition Phase dauert 2 Wochen. Diese Zeit wird verwendet um alle Dokumente fertigzustellen und das Projekt ordnungsgemäss zu beenden.



A.6 Milestones

A.6.1 MS1: Projektübersicht erstellt

Datum	02. März 2018
Ziele	Erste Version des Projektplans Allgemeine Übersicht über das Projekt Erweiterungsanalyse durchgeführt Arbeitspakete grob definiert
Dokumente	Projektplan inkl. Erweiterungsanalyse & Arbeitspaketen
Verfügbare Zeit	2 Wochen: $2 * 2 * 20h = 80h$

A.6.2 MS2: Analyse durchgeführt

Datum	16. März 2018
Ziele	Anforderungen festgelegt (Dokumentiert das WAS): • Use-Case-Diagramm erstellt • Anforderungen an das System aufgelistet Domainanalyse durchgeführt, die Domainanalyse beinhaltet: • Eine Analyse der möglichen Tools für die Erweiterung von OATS Nichtfunktionale Anforderungen definiert (Dokumentiert das WIE)
Dokumente	Domainanalyse Anforderungsspezifikation
Verfügbare Zeit	2 Wochen: $2 * 2 * 20h = 80h$

A.6.3 MS3: End of Elaboration

Datum	30. März 2018
Ziele	Definition der neuen Architektur von OATS, mit folgenden Inhalten: • Systemübersicht: beschreibt die Komponenten des Systems • Tools: stellt die im System eingesetzten Tools vor • Schnittstellen: beschreiben der Schnittstellen inner- und ausserhalb des Systems. Beschreibt auch die wichtigsten Abläufe im System. • Datenmodellierung: zeigt auf, wie die Daten im System dargestellt bzw. abgelegt werden Prototyp für die Erweiterungen erstellt, dies beinhaltet : • Testumgebung mit den Erweiterungen aufgestellt
Dokumente	Präsentation Prototyp (nicht als Dokument, live mit den Betreuern) Architekturdokument
Verfügbare Zeit	2 Wochen: $2 * 2 * 20h = 80h$

A.6.4 MS4: Neue Features Teil 1, implementiert

Datum	13. April 2018
Ziele	Migrieren der Netzwerkdaten in Netbox Anbinden Streaming-Telemetry an OATS - Anbinden Timeseries-database - Anbinden Kafka - Entwickeln von Streaming-Telemetry-Client - Entwickeln von einfachem Workflow
Dokumente	Zwischenresultate Präsentation der implementierten Funktionen
Verfügbare Zeit	2 Wochen: $2 * 2 * 20h = 80h$



A.6.5 MS5: Neue Features Teil 2, implementiert

Datum	27. April 2018
Ziele	Code Refactoring - Datenformat napalm-logs anpassen - Napalm-logs Client verbessern - Evtl. salt client Deployment-Verbesserungen Workflow entwickeln mit Netflow und Streaming Telemetry
Dokumente	Zwischenresultate Präsentation der implementierten Funktionen
Verfügbare Zeit	2 Wochen: 2 * 2 * 20h = 80h

A.6.6 MS6: Neue Features Teil 3, implementiert

Datum	11. Mai 2018
Ziele	Vereinfachung der Implementierung von neuen Workflows Standardisierung der Events von verschiedenen Quellen Implementieren von Netflow Client Konzept zur Konfiguration mittels zentraler Datei erstellen
Dokumente	Zwischenresultate Präsentation der implementierten Funktionen
Verfügbare Zeit	2 Wochen: 2 * 2 * 20h = 80h

A.6.7 MS7: Neue Features Teil 4, implementiert

Datum	25. Mai 2018
Ziele	Dashboard entwickeln Weitere Workflows umsetzen API entwickeln Logging einrichten
Dokumente	Zwischenresultate Präsentation der implementierten Funktionen
Verfügbare Zeit	2 Wochen: 2 * 2 * 20h = 80h

A.6.8 MS8: End of Construction

Datum	06. Juni 2018
Ziele	Funktionalität des Systems stimmt mit Anforderungen überein, die in der Elaboration definiert wurden Code cleanup
Dokumente	Umgesetzte Anforderungen werden in die Doku übertragen
Verfügbare Zeit	1 Wochen: 1 * 2 * 20h = 40h

A.6.9 Abgabe

Datum	15. Juni 2018
Beschreibung	Erstellung Abstract & Poster Alle Dokumente abgegeben
Dokumente	Alle für die Bewertung relevanten Dokumente
Verfügbare Zeit	2 Wochen: 2 * 2 * 40h = 160h



A.6.10 Besprechungen

- Die offiziellen Besprechungen des Teams mit den Betreuern finden jeweils freitags um 13:00 im Gebäude 8 der HSR statt. Dabei werden die Resultate der vorigen Woche und offene Fragen des Projektteams diskutiert.
- Das Team sendet den Betreuern am Donnerstag jeweils die relevanten Dokumente für die Besprechungen am Freitag.

A.6.11 Arbeitsabläufe

Das Team trifft sich offiziell an 3 Tagen pro Woche:

- Montag:
 - 08:00 – 15:00: Gemeinsames Arbeiten am Projekt
- Donnerstag:
 - 08:00 – 10:00: Gemeinsames Arbeiten am Projekt
 - 12:00 – 15:00: Gemeinsames Arbeiten am Projekt
- Freitag:
 - 08:00 – 13:00: Gemeinsames Arbeiten am Projekt
 - 13:00 Besprechung mit den Betreuern
 - 14:00 – 16:00: Nachbesprechung und erstellen des Protokolls

Dies ergibt ca. eine gemeinsame Arbeitszeit von 18-22 Stunden. Die restliche Arbeitszeit wird von den Teammitgliedern frei auf die Woche verteilt, sodass die 20 Stunden pro Woche im Schnitt mindestens erreicht werden.



A.7 Arbeitspakete

A.7.1 Elaboration: MS1, MS2, MS3

Es existieren einige Arbeitspakete bei denen keine Zeit geschätzt und geschrieben wurde. Diese Pakete wurden zu Beginn erstellt, wurden aber entweder nicht umgesetzt, oder deren Zeit wurde zusammen mit anderen Paketen aufgeschrieben.

Nr.	Name	Inhalt	Soll [h]	Ist [h]
1	Analyse möglicher Korrelation Tools	Es existieren bereits Tools welche Daten auswerten und korrelieren können, diese sollen analysiert und verglichen werden um das Beste für unser System zu identifizieren.	20	9.5
2	Korrelation von Events	Das System kann erhaltene Events korrelieren und zusammenfassen.	2	1
3	Korrelierte Events als Ganzes lösen	Die zusammengefassten Events sollen als Ganzes gelöst werden können.	0	0
4	Analyse möglicher Telemetrie Daten Tools	Es existieren bereits Tools welche Telemetrie Daten aus einem Netzwerk erhalten und verarbeiten können, diese sollen analysiert und verglichen werden um das Beste für unser System zu identifizieren.	12	16.5
5	Telemetrie Daten erhalten	Das System kann, durch Verwendung eines Open Source Tools, Telemetrie Daten aus einem Netzwerk erhalten.	8	5.5
6	Telemetrie Daten verarbeiten	Das System kann die erhaltenen Telemetrie Daten auswerten und mögliche Lösungen auswählen.	0	0
7	Telemetry Lösung deployen	Das System kann, basierend auf der Auswertung der Telemetrie Daten, Lösungen deployen.	8	12.75
8	Analyse Netzwerk Inventar Tool	Zur Erstellung eines Netzwerkinventars mit allen Geräten gibt es bereits diverse Tools, diese sollen analysiert und verglichen werden um das Beste für unser System zu identifizieren	8	5.5
9	Implementierung Netzwerk Inventar Tool	Sobald ein ideales Tool gefunden wurde gilt es dies aufzusetzen.	0	0
10	Automatisches Erstellen der Datenbank	Die Daten welche mit dem Tool zum Erstellen des Netzwerkinventars erhalten wurden, sollen dazu verwendet werden, um direkt die Datenbank anzulegen.	0	0
11	Analyse besserer Datenbanken	Analysieren und vergleichen verschiedener Open Source Datenbanklösungen.	8	15
12	Umstellung der Datenbank	Die alte Datenbank durch die am besten geeignete Datenbank ersetzen.	10	7
13	Aggregation Implementierung auf Logs	Die Logeinträge die zurzeit in die Datenbank gespeichert werden, haben zurzeit keinen Nutzen. Mit einem Konzept zur könnte man dies verbessern.	0	0



14	Entwurf eines Refactoring Konzepts	Es muss entschieden werden, in welcher Sprache (z.B. YAML) die externen Files geschrieben werden sollen. Zudem ist es sinnvoll schon im Voraus ein sinnvolles Konzept zur Umsetzung vorzubereiten.	8	3
15	Code-Refactoring	Der Code muss so angepasst werden, dass das Erweitern via externem File möglich wird.	4	3.5
16	Deployment-Analyse	Es muss analysiert werden, inwiefern ein vereinfachtes Deployment erreicht werden kann. Es besteht die Möglichkeit eines einzelnen Scripts, bis zur kompletten Verpackung in Docker Container von OATS.	6	3.5
17	Realisierung des Deployments	Durchführen der in der Analyse festgelegten Massnahmen.	0	0
18	Analyse möglicher GUI-Technologien	Es existieren mehrere Arten von GUI (Website, Desktop-App). Es muss ermittelt werden welche Art die Beste ist und welche Technologien für die Umsetzung in Frage kommen.	6	6
19	Designentwürfe	Erstellen von ersten GUI-Entwürfen.	6	0
20	Darstellen des aktuellen Netzwerk-Zustands	Der aktuelle Zustand des Netzwerks wird im GUI dargestellt.	0	0
21	Editieren der OATS-Datenbank	Über das GUI können Datenbankeinträge editiert werden (z.B. eine Änderung im Netz, behandeln von Problemfällen).	0	0
22	Analyse möglicher Sensoren	Es existieren verschiedene Sensoren, welche Events auslösen können. Es soll analysiert werden, welche für OATs in Frage kommen.	4	8.25
23	Einbinden der Alternativen ins System	Die Syslog-Alternative(n) muss in OATS eingebunden werden.	0	0
24	Überarbeitung Domainanalyse	Die Domainanalyse muss überarbeitet werden	4	20.25
25	Überarbeitung Anforderungen	Die Anforderungsspezifikation muss überarbeitet werden.	6	16
26	Sitzungen	Durchführen der Sitzungen	8	8.25
27	Allgemeines Proj.-mgmt	Allgemeines Projekt-Management	40	50.5
28	Troubleshooting-Fälle erfassen	Erfassen von Troubleshooting-Fällen zusammen mit U. Baumann und L. Metzger	8	5
29	Überarbeitung Architekturdoku	Überarbeitung der Architekturdoku	16	22.5
Total			192	219.5



A.7.2 Construction: Milestone 4

Nr.	Name	Inhalt	Soll [h]	Ist [h]
1	Migririen der Netzwerkdaten	Die Netzwerkdaten müssen von der MongoDB in die netbox migriert werden.	5	4
2	API für netbox Daten	API für Netbox entwickeln	15	14
3	API für postgres Daten	API für Postgres entwickeln	10	13
4	Anbinden InfluxDB	Anbinden von InfluxDB ans System	5	5
5	Anbinden Kafka	Anbinden von Kafka ans System	4	0
6	Streaming-Telemetry Client	Streaming-Telemetry Client entwickeln (netconf)	6	4.75
7	Stream Processing Client (Java)	Stream Processing Client entwickeln (Kafka Streams)	12	36.5
8	Netzwerkdaten Simulation (Iperf)	Konzept zur Simulation der Netzwerkdaten entwickeln	8	0
9	Erster Workflow implementieren	Erster Workflow mit Streaming-Telemetry-Daten implementieren	16	6
10	Projekt mgmt	Allgemeines Projekt-Management	2	1.5
		Total	83	84.75



A.7.3 Construction: Milestone 5

Nr.	Name	Inhalt	Soll [h]	Ist [h]
1	Napalm-Logs Client anpassen	Napalm-Logs Client an neue OATS-Umgebung anpassen (Kafka)	12	6.5
2	CI aufsetzen	CI aufsetzen	6	5
3	Skripts daemonisieren (Systemd)	Daemonisieren der einzelnen OATS-Prozesse aufsetzen	10	6
4	Dockerisieren der Clients/Applikationen	Dockerisierung der einzelnen Prozesse	20	12
5	Code Cleanup & Testing	Bestehender Code aufräumen und Testing verbessern	18	26.75
6	Workflow umsetzen	Umsetzen eine neuen Workflows (Port-Flapping)	8	12
7	Präsentation	Vorbereiten/Durchführen der Zwischenpräsentation	6	10.25
8	Projekt mgmt	Allgemeines Projekt-Management	2	3.75
		Total	82	82.25



A.7.4 Construction: Milestone 6

Nr.	Name	Inhalt	Soll [h]	Ist [h]
1	Events Standardisieren: Konzept	Events in OATS sollen einheitlich sein, unabhängig vom Sensortyp	2	1
2	Events Standardisieren: Umsetzung	Umsetzung des Event-Standarts	6	26.25
3	Netflow implementieren	Einfügen von Netflow in OATS	18	22.25
4	Flask-basierte API implementieren	Entwickeln einer Flask-basierten API für OATS	8	9
5	einheitliches directory für OATS erstellen	z.b. /etc/oats/....	4	2.5
6	Einheitliche Konfig-Datei erstellen	Erstellen einer einheitlichen Konfigurationsdatei für OATS	4	3.5
7	Konfig-Datei in Kafka-Streams auslesen	Auslesen der Datei	16	13.25
8	Datenbank migrieren	Migration der Datenbank von MongoDB zu PSQL/InfluxDB	2	7
9	Projekt mgmt	Allgemeines Projekt-Management	2	4
10	Neue Datenbank Funktionen für Workflows	Entwickeln neuer Datenbankfunktionen für die alten und neuen Workflows	16	16
		Total	78	104.8



A.7.5 Construction: Milestone 7

Nr.	Name	Inhalt	Soll [h]	Ist [h]
1	Grafana einbinden	Grafana einbinden, um die Daten in OATS zu visualisieren	12	13.5
2	InfluxDB read implementieren	Funktion für Telemetry-Workflow entwickeln	10	12
3	Port-Flapping Workflow	Port-Flapping Workflow implementieren	12	5.5
4	langsame Workflows wegen langsamen napalm Zugriffen	Bugfix	3	1.5
5	Flask API überarbeiten	Allgemeines Überarbeiten der Flask-API	5	8
6	config für port drosseln auf Gerät laden	Config für Telemetry-Workflow aufs Gerät laden	12	15.25
7	Korrelation ohne kafka-streams ermöglichen	Überarbeitung der Konfig-Datei	4	4
8	Flask basiertes UI entwickeln	Event erstellen über UI ermöglichen	8	11
9	Logging einrichten	Allgemeines Logging im Code für einfacheres Debuggen der Daemon-Prozesse	10	9.5
10	Projekt mgmt	Allgemeines Projekt-Management	2	2.5
Total			78	82.75



A.7.6 Construction: Milestone 8

Nr.	Name	Inhalt	Soll [h]	Ist [h]
1	Excel Zeiterfassung in influx für grafana	Skript schreiben um Excel Daten in die InfluxDB zu schreiben	6	9.5
2	Vagrant Umgebung	Aufsetzen einer Vagrant-Box inkl. OATS-install	12	8.5
3	Refactoring	Allgemeines Cocde-Refactoring	8	10
4	Code dokumentation	Allgemeine Code-Dokumentation	8	7
5	Testing und Exception Handling	Allgemeines Testing & Exception-Handling	8	9
6	Workflow mit Config	Telemetrie-Workflow abschliessen	2	1
7	Abstract schreiben	Schreiben des Abstracts	2	2.75
8	requirements.txt	Aufsetzen einer Requirements-Datei mit welcher Python-Packages für OATS installiert werden können	1	0
9	Projekt mgmt	Allgemeines Projekt-Management	2	3
Total			49	50.75



A.7.7 Transition: Abgabe

Nr.	Name	Inhalt	Soll [h]	Ist [h]
1	Domainanalyse	Überarbeiten der Domainanalyse	4	4
2	Architekturdoku	Überarbeiten der Architekturdoku	20	28.75
3	Management Summary	Schreiben Management-Summary	2	1
4	Technischer Bericht	Schreiben des technischen Berichts	6	8.5
5	Anforderungsdoku	Überarbeiten der Anforderungsdoku	4	4
6	Anleitungen	Schreiben der verschiedenen Anleitungen (Installation/ Konfiguration/ Development)	10	12
7	Persönliche Reflektion	Schreiben der persönlichen Reflektion	2	2
8	Projektplan	Überarbeiten/Updaten des Projektplans	4	5
9	Projekt mgmt	Allgemeines Projekt-Management	2	3.5
10	Abstract	Überarbeiten des Abstracts	2	2
11	Präsentation vorbereiten	Vorbereiten der Präsentation für die Abschlussprüfung	12	12
12	Poster	Erstellen des Posters für die Ausstellung	4	6.5
13	Review Dokumente	Allgemeines Review der Dokumente	20	26
14	Vorbereitung Abgabe	Vorbereiten der Dokumente für die Abgabe	10	18
15	Ausstellung	Abschluss-Ausstellung der BAs	10	10
Total			112	143.3



A.8 Risikomanagement

A.8.1 Eingetretene Risiken in der Studienarbeit

In der Studienarbeit sind zwei der erstellten Risiken eingetreten, diese wurden erneut überprüft und zusätzlich Zeit und Massnahmen sind in die neue Risikoanalyse eingeflossen. Hauptsächlich basiert die Risikoanalyse auf den Risiken welche auch in der Studienarbeit vorhanden waren das sich der Themenbereich nicht geändert hat und dies eine Folgearbeit ist.

A.8.2 Risiken

Die Risikoplanung ist im Anhang unter dem Anhang E (Technische Risiken) zu finden.

Auf der ersten Seite sind folgende Informationen zu finden:

- Risikobeschreibung
- Max. Schaden: Geschätzter Zeitverlust, sollte das Risiko eintreten
- Eintrittswahrscheinlichkeit: Geschätzte Wahrscheinlichkeit, dass das Risiko eintritt
- Gewichteter Schaden: Schaden der aufgrund von eingetretenen Risiken zusätzlich aufgewendet werden muss
- Verhalten beim Eintreten: Massnahmen die ergriffen werden, sollte das Risiko eintreten

Je nach dem zu welchem Zeitpunkt ein Risiko eintritt, wird der Schaden grösser oder kleiner. Der gewichtete Schaden berücksichtigt dies.

Auf der zweiten Seite werden die eingetretenen Risiken erfasst.

A.8.3 Umgang mit Risiken

Sollten Probleme auftreten, wird die dafür aufgewendete Zeit im Risikodokument niedergeschrieben. Die Zeit wird entweder als Verlust (gegen Ende des Projekts nicht anders möglich) akzeptiert, oder wird durch zusätzlichen Mehraufwand des Projektteams kompensiert.

A.9 Infrastruktur

A.9.1 Entwicklungswerkzeuge

- PyCharm¹: für Python-Code
- Vim²: zum Editieren von Text direkt auf dem OATS Server
- Microsoft Office³: Excel & Word, zum Erstellen der Dokumentation & Zeiterfassung
- Git/Github⁴: Versionskontrolle
- Astah⁵: zum Erstellen von UMLs

¹ <https://www.jetbrains.com/pycharm/>

² <http://www.vim.org/>

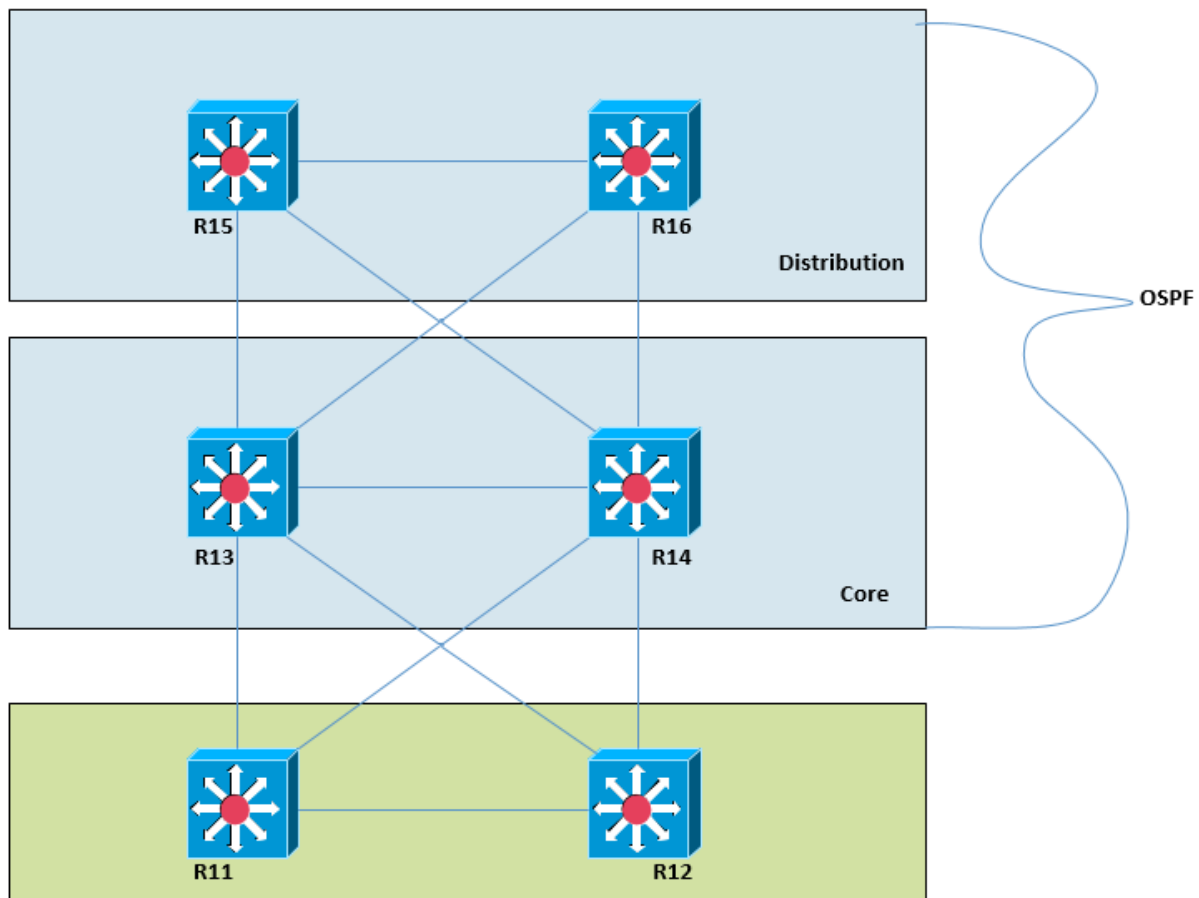
³ <https://products.office.com/de-ch/home>

⁴ <https://github.com/>

⁵ <http://astah.net/de>



A.9.2 Testumgebung



Folgendes Netz wurde durch die Betreuer aufgesetzt und durch das Projektteam für die Entwicklung von OATS, bereits während der Studienarbeit benutzt

Die Router befinden sich alle im gleichen Subnetz. Zudem sind alle Geräte über ein Management-Netz erreichbar, in welchem sich auch der Server auf dem OATS läuft befindet.

A.9.3 Testing

Für diese Bachelorarbeit wird auf einen offiziellen Systemtest verzichtet, da das System dafür ausgelegt ist, von gut ausgebildeten Personen benutzt zu werden und da der in der Studienarbeit durchgeführte Systemtest zu wenig Nutzen erbracht hatte.

Der wohl sinnvollste Test, den man OATS unterziehen könnte, wäre das Deployment in einem realen Netzwerk. Dies ist im Rahmen der Bachelorarbeit leider nicht möglich, wird aber als Möglichkeit für die Zukunft notiert.

Die einzelnen Teilsysteme werden, sofern sinnvoll, durch Unit-Tests abgedeckt. So werden alle Teilsysteme welche nicht nur zum Durchreichen von Daten benötigt werden, getestet. Dies betrifft vor allem die Datenbankzugriffe und der Umgang mit der zentralen OATS Konfigurationsdatei.

A.10 Qualitätsmassnahmen

A.10.1 Dokumentation

Sämtliche Dokumente werden in ein eigenes Git-Repository (separat vom Code) eingchecked und deren Qualität wird durch Peer-Reviews sichergestellt.



A.10.2 Zeiterfassung / Arbeitspakete

Es wurde bewusst kein umfassendes Tool wie Redmine oder Jira benutzt um die Zeiterfassung bzw. die Arbeitspakete zu verwalten. Dies wurde direkt in Excel erledigt.

Der Grund dafür liegt im grossen Zeitaufwand der benötigt wird um ein solches Tool aufzusetzen, der sich unserer Meinung für 2 Teammitglieder nicht lohnt. Vor allem weil sich das Team täglich trifft und dadurch immer über das direkte Gespräch über den aktuellen Stand des Projekts informiert wird.

A.10.3 Coding Guidelines / Entwicklung

Als Coding Guideline während der Entwicklung wurde der in PyCharm integrierte Coding Guide PEP8 verwendet.



Anhang B: Technische Risiken

B.1 Risikomanagement

Projekt:	Automated Troubleshooting	Dieses Dokument beschreibt die spezifischen Projektrisiken, welche bei der Durchführung auftreten könnten.					
Erstellt am:	26.02.2018	Allgemeine Risiken (wie Krankheit, Unfall etc.) werden hier nicht abgebildet. Der gewichtete Schaden, ist die aufgrund von eingetretenen Risiken zusätzliche erwartete Zeit die aufgewendet werden muss. Er wird am Projektende mit dem eingetretenen Schaden verglichen.					
Autor:	Raphael Jöhl						
Gewichteter Schaden:	27.95						
Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten
R1	Inkompatible Tools	Verschiedene Tools sind nicht miteinander kompatibel	25	20%	5	Gründliche Recherche der Tools inklusive Details wie z.B. verwendete Datenformate.	Inkompatible Komponenten mit existierenden Tools oder einer Eigenlösung ersetzen
R2	Fehlerhaftes/nicht verstandenes Tool	Ein Tool welches ausgewählt wurde, funktioniert nicht wie erwartet, wurde falsch verstanden oder ist fehlerhaft	24	20%	4.8	Tools bevorzugen, die schon längere Zeit im Einsatz sind und über eine grosse Community verfügen. Stable-Releases verwenden, keine Beta-Versionen.	Überprüfen ob das Problem in akzeptabler Zeit von den Entwicklern behoben werden kann. Sonst wechseln von Tools bzw. Entwicklung einer Eigenlösung.



R3	Schlechte Kommunikation	Die Kommunikation zwischen den Projektmitgliedern oder mit den Betreuern ist schlecht, was zu Fehlern bei der Arbeit führt.	15	25%	3.75	Besprechungen protokollieren, regelmässige Besprechungen zwischen den Teammitgliedern. Dieses Risiko ist in der Studienarbeit aufgetreten, weshalb wir hier besonders acht geben dies zu verhindern.	Fehler beheben und Kommunikation verbessern.
R4	Dokumente- / Datenverlust	Daten werden unbewusst gelöscht, überschrieben	4	10%	0.4	Einsatz von Versionierungstools	Wiederherstellen des Backups, Commit
R5	Wechselnde Anforderungen	Neue oder ständig wechselnde Anforderungen	30	5%	1.5	Klare Abgrenzung des Scopes und gute Planung	Anforderungen analysieren und allenfalls einplanen
R6	Fehlerhafte dependencies	Dependencies, auf die das System aufbaut sind fehlerhaft.	30	25%	7.5	Recherche der dependencies, stable Versionen benutzen. Dieses Risiko ist in der Studienarbeit aufgetreten, da wir hier aber auf externe , können wir dieses Risiko nicht direkt verhindern.	Rücksprache mit Entwicklern der dependency um Fehler zu beheben.
R7	Anforderungen sind komplexer als angenommen	Komplexität wurde unterschätzt. Der angenommene Zeitaufwand	50	10%	5	Gute Recherche	Rücksprache mit Betreuer und Team über weiteres Vorgehen, evtl.



		übersteigt die Planung um ein vielfaches.					verringern des Funktionsumfangs
Summe			178		27.95		

B.2 Eingetretene Risiken

Risiko No.	Beschreibung	Grund	Zusätzlich aufgewendete Zeit [h]	Weiteres Vorgehen
1	Cisco Gerät liefert nicht alle angeforderten Daten	BGP YANG Modell ist auf IOS-XE nur teilweise implementiert.	8.00	Wurde kompensiert, indem ein anderer Workflow implementiert wurde

