Höhere Technische Bundeslehranstalt Wien 3, Rennweg
Rennweg 89b
A-1030 Wien, Tel +43 1 24215-10

Diplomarbeit

Nexus
Traffic-Analyse, Datenbereitstellung

ausgeführt an der
Höheren Abteilung für
Informationstechnologie/Netzwerktechnik

im Schuljahr 2020/2021

durch
Lukas Kaiser
Patrick Krenn
Simon Seiler

unter der Anleitung von

DI Christian Schöndorfer

DI Clemens Kussbach

DI Robert Dazinger

Wien, März 2021

Kurzfassung

Netzwerkadministratoren kann es schwerfallen, den Überblick über ihr Netzwerk zu behalten. Am Beispiel der HTL Rennweg wird klar, dass es sich bei mehr als tausend Endgeräten um eine enorme Menge an unübersichtlichem Netzwerkverkehr handelt. Mittels Deep Packet Inspection (DPI) können Netzwerkpakete genauer inspiziert werden, um Rückschlüsse auf Netzwerknutzung und aktive Anwendungen ziehen zu können. Dies kann in dieser Diplomarbeit nutzbringend verwendet werden.

Das Ziel dieser Diplomarbeit ist es, den internen sowie den von der Schule ausgehenden Netzwerkverkehr zu analysieren. Im Rahmen dessen wird die Erfassung der Datenmengen, Erkennung von Portscans, Erkennung von Netzwerkverbindungen und die Kategorisierung dieser angestrebt. Die daraus resultierenden Daten werden zentralisiert gespeichert und zu Statistiken weiterverarbeitet. Mittels API werden die Daten weiters für die externe Nutzung zur Verfügung gestellt.

Am Ende der Diplomarbeit hat sich besonders eines herausgestellt: Das Inspizieren der Daten ist enorm von der Rechenleistung der auswertenden „Maschinen“ abhängig. In denkbar kleineren Testumgebungen funktioniert das Analysieren der verschiedenen Daten einwandfrei. Doch dem Erfassen und Analysieren der Daten einer gesamten Schule in Echtzeit ist der Großteil der Skripte nicht gewachsen, da die Effizienz der Skripte und die Ressourcen der „Maschinen“ in „Stoßzeiten“ meist nicht ausreichen. Hingegen stellt das grundlegende Erlangen der Daten eine geringere Hürde dar. Weiters ist das schlichte Erfassen der Datenmenge sehr gut und effizient umsetzbar, da Pakete, anders als bei der Analyse, nicht weiter inspizieret werden müssen.

Den vorliegenden Ergebnissen kann entnommen werden, dass es schwierig ist, mithilfe selbst geschriebener Skripte den Netzwerkverkehr einer Schule dieser Größe genau zu untersuchen. Bei Schulen mit schlechterer Infrastruktur, also mit weniger Ressourcen, die zum Ausführen der Skripte zur Verfügung stehen, und Netzwerkgeräten, die Probleme bei der Verarbeitung oder Weiterleitung der Pakete haben, würde dies noch stärker zum Verhängnis werden. Es müssten entweder bessere Geräte angeschafft werden oder die Genauigkeit der analysierten Daten müsste darunter leiden. Mithilfe dieses Buches wird ein möglicher Weg zur Analyse eines Netzwerkes aufgezeigt.

Abstract

Network administrators might face barriers to keeping track of their whole network. Illustrated by the example of the HTL Rennweg, it becomes clear that with more than a thousand end devices there is an enormous, big amount of confusingly complex network traffic. Utilizing Deep Packet Inspection (DPI), network packets can be inspected more precisely, and various information can be obtained from them. This can be put to good use.

The aim of this diploma thesis is to analyse the internal as well as the outgoing network traffic of the school. Within the scope of this, the collection of data volumes, the detection of port scans, the detection of network connections, and the categorization of these are aimed at. The resulting data are stored centrally and processed into statistics. Furthermore, the data are made available for external use via an API.

At the end of the diploma thesis, one thing, in particular, became clear: The inspection of the data is enormously dependent on the computing power of the evaluating machines. In conceivably smaller test environments, analysing the various data works perfectly fine. However, the majority of the analysing scripts cannot cope with the data volume of an entire school. The basic acquisition of data is less of a hurdle. What is more, the simpler acquisition of the amount of data is very well and efficiently realizable due to the aspect of not having to look further into network packets.

From the results, it can be seen that it is difficult to accurately examine the network traffic of a school of this size with the help of self-written scripts. For schools with poorer infrastructure, this would be even more of a challenge. With the help of this book, a possible way to analyse a network is shown.

Ehrenwörtliche Erklärung

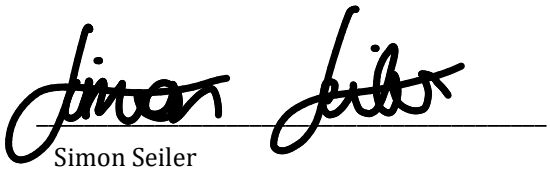
Ich versichere,

- dass ich meinen Anteil an dieser Diplomarbeit selbstständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe
- und mich auch sonst keiner unerlaubten Hilfe bzw. Hilfsmittel bedient habe.

Wien, am 01.05.2021

Lukas Kaiser

Patrick Krenn



Simon Seiler

Präambel

Die Inhalte dieser Diplomarbeit entsprechen den Qualitätsnormen für „Ingenieurprojekte“ gemäß § 29 der Verordnung des Bundesministers für Unterricht und kulturelle Angelegenheiten über die Reife- und Diplomprüfung in den berufsbildenden höheren Schulen, BGBl. Nr. 847/1992, in der Fassung der Verordnungen BGBl. Nr. 269/1993, Nr. 467/1996 und BGBl. II Nr. 123/97.

Liste der betreuenden Lehrer
Professor, Diplom-Ingenieur, Christian Schöndorfer
Professor, Diplom-Ingenieur, Clemens Kussbach
Professor, Diplom-Ingenieur, Robert Dazinger

Liste der Kooperationspartner:
Keine vorhandenen Kooperationspartner

Inhaltsverzeichnis

1 Ziele

Für Netzwerkadministratoren stellt sich immer wieder eine Frage: „Wie kann ich den dauerhaften Überblick meines ständig wachsenden Netzwerks gewährleisten?“ Zwar gibt es die Möglichkeit, auf kommerzielle Produkte oder auf Lösungen anderer Firmen zu setzen, jedoch sind diese meist mit einem kostspieligen Aufwand verbunden, da hier oft Lizenzen angeschafft werden müssen. Dies ist vor allem aus der Sicht einer Schule schwierig zu vertreten.

Das Ziel dieser Diplomarbeit ist es, gemeinsam mit zwei anderen Diplomarbeiten („Cerebrum“ und „Oculus“), dem Administrator einen Ausweg aus der oben geschilderten Lage zu ermöglichen. Hierfür soll einerseits der Traffic des Schulnetzwerks analysiert und die daraus resultierenden Daten zentralisiert gespeichert werden. Neben der von dieser Diplomarbeit aufbereiteten Daten werden auch die der von der Diplomarbeit „Cerebrum“ erarbeiteten Daten aufbewahrt. All die gesammelten Daten werden zu Statistiken verarbeitet und werden via API in unverarbeiteter und verarbeiteter Form der Diplomarbeit „Oculus“ zur Verfügung gestellt.

1.1 Individuelle Zielsetzung

Im Rahmen einer Diplomarbeit ist es notwendig, jedes Ziel einem Teammitglied zuzuordnen. Deshalb ist im Folgenden die Zuordnung der Ziele gemäß dem Antrag der Diplomarbeit festgehalten:

1.1.1 Hauptziele

Bei den Hauptzielen handelt es sich um die wichtigsten Ziele der Diplomarbeit. Beim Nichterreichen einer dieser Ziele kann die Diplomarbeit als gescheitert gewertet werden.

Planung der Datenstruktur (Ziel H1, Krenn)
Für die Speicherung der gewonnen Daten sowie der Statistiken ist eine entsprechende Datenstruktur ausgewählt.

Entsprechend der massiven Menge von Daten wird eine geeignete nicht-relationale Datenbank ausgewählt. Anhand der von den Teammitgliedern gelieferten Daten werden entsprechende Strukturen geplant.

Weiters wird eine geeignete relationale Datenbank für die Speicherung der Statistiken ausgewählt. Die Speicherung dieser wird anhand von Star-Schemen geplant.

Erstellung der Datenstruktur (Ziel H2, Krenn)
Anhand der erarbeiteten Planung sind die beiden Datenbanken erstellt. Weiters besteht ein Python-Skript je Datenbank.

Anhand der festgehaltenen Planung wird eine Datenbank für die Speicherung der Ursprungsdaten aufgesetzt. Zusätzlich wird ein Skript (z.B. in Python) zum Beschreiben dieser Datenbank erstellt. Dabei wird die zuvor erarbeitete Datenstruktur verwendet.

Wartung der Datenstruktur (Ziel H3, Krenn)
Die Datenstruktur ist in Betrieb und entspricht performancetechnisch den Anforderungen.

Die Datenbank wird im Laufe des Projekts ständigen Audits unterzogen und geprüft. Kommt es dabei zu schlechten Resultaten, wird die Datenstruktur überarbeitet. Entsprechend der im Projekt neu entstandenen Anforderungen werden Änderungen an der Speicherstruktur vorgenommen.

Erstellung von Statistiken (Ziel H4, Krenn)
Eine entsprechende Datenbank ist für die Speicherung der Tagesstatistiken gemäß der Planung erstellt. Am Ende eines Tages sind die Statistiken erstellt.

Die zur Speicherung von Tagesstatistiken verwendete Datenbank wird aufgesetzt. An einem später festgelegten Zeitpunkt eines Tages werden die für Statistiken benötigten Daten abgefragt. Anschließend werden sie entsprechend aufbereitet und in der dafür vorgesehenen Struktur gespeichert.

Erfassung der quantitativen Datenmenge eines Klassenraums (Ziel H5, Kaiser)
Von zumindest einer Beispiel-Schulklasse ist mittels SNMP-Manager periodisch die verbrauchte Datenmenge über den verbundenen Switch abgefragt und in der erstellten Datenstruktur abgespeichert.

Der SNMP-Manager fragt periodisch den mit der Schulklasse verbunden Switch ab. Der Switch ist mit entsprechendem Community-String und Passwort konfiguriert. Nach Abfrage durch den SNMP-Manager schickt der Switch dem SNMP-Manager die abgefragten Daten. Mittels eines Skripts werden die Daten in einer passenden Struktur gespeichert und der Datenstruktur übergeben.

Erkennung von Portscans (Ziel H6, Kaiser)
Einfache Portscans auf einer ausgewählten Firewall sowie auf einem ausgewählten EDV-Switch werden erkannt und in der Datenstruktur abgespeichert.

In einer Testumgebung werden im ersten Schritt Portscans beobachtet. Auf diese Weise werden Besonderheiten beziehungsweise Erkennungsmerkmale dieser erfasst. Mithilfe dieser wird ein Muster eines einfachen Portscans erstellt, um diese identifizieren zu können.

Praktische Umsetzung Portscans (Ziel H7, Kaiser)
Unter Verwendung des erstellten Musters werden Portscans im Netzwerkverkehr einer Klasse erkannt.

Zur Analyse des Netzwerkverkehrs wird die Packet Inspection Infrastruktur des Partnerteams „Cerebrum“ verwendet. Beide Analysemöglichkeiten werden zur Erkennung von Portscans getestet. Anschließend wird eine Entscheidung getroffen, um die bestmögliche Leistung für den Produktivbetrieb zu gewährleisten.

Erstellung des API-Konzepts (Ziel H8, Kaiser)
Nach Absprache mit dem Oculus-Team sind entsprechende Anforderungen festgehalten. Entsprechend dieser ist ein Konzept erstellt.

Es wird eine entsprechende Schnittstelle zwischen der Datenstruktur und der Website von Team Oculus geplant und festgehalten. Anforderungen von Team Oculus an die Daten werden berücksichtigt. Das Versenden von doppelten Datensätzen muss gehandhabt werden. Dabei wird es sich um eine private API innerhalb des Schulnetzwerkes handeln.

Erstellung der API (Ziel H9, Kaiser)
Anhand des erstellten Konzepts ist die API umgesetzt.

Die API wird nach Konzept umgesetzt und stellt die Schnittstelle zwischen der Datenstruktur und der Website von Team Oculus dar. Dabei wird ein Authentifizierungsmethode konfiguriert, um ungewollten Zugriff zu verhindern. Mittels daemon wird dafür gesorgt, dass die API automatisch aktiv ist. Um Abstürze zu verhindern, wird Error-Handling integriert.

Erfassung aktiver Netzwerkverbindungen eines Klassenraums (Ziel H10, Seiler)
Relevante Netzwerkverbindungen (TCP, UDP, ICMP) eines Klassenraums sind in einem Collector gesammelt und in einer Datenstruktur abgespeichert.

Von mindestens einer Beispiel-Schulklasse werden alle ein- und ausgehenden TCP-, UDP- und ICMP-Verbindungen mittels dem Traffic-Analyzer NetFlow erfasst und in dem einzurichtenden NetFlow-Collector gespeichert. Die gesammelten Daten werden aus dem NetFlow-Collector mittels eines Skripts der Datenstruktur übergeben.

Erstellung zeitlich geplanter Datenübergabe mittels Cronjobs (Ziel H11, Seiler)
Das Speichern der gesammelten Daten ist durch zeitlich geplantes Ausführen der individuell erstellten Skripts mittels Cronjobs umgesetzt.

Ein Entwurf eines Zeitplans für das Speichern der individuell gespeicherten Daten in die Datenstruktur wird erstellt. Nach diesem Entwurf werden die individuell erstellten Skripts mittels Cronjobs aufgerufen, um die Übergabe der Daten zeitlich zu managen.

Management des DA-Teams (Ziel H12, Seiler)
Das Projekt wird, hinsichtlich des Managements, ordnungsgemäß mittels einer zielgerechten Projektmanagement-Methode umgesetzt und dokumentiert.

Sinngemäß der Projektmanagement-Methode werden entsprechende Planungs- und Controlling-Dokumente verfasst. Weiters werden regelmäßig interne Besprechungen abgehalten und Meetings mit dem Betreuerteam durchgeführt. Zusätzlich steht der Projektleiter im ständigen Austausch mit den Projektleitern der Partnerteams Cerebrum und Oculus, um über deren Fortschritt im Bilde zu sein. Während der gesamten Projektdauer wird versucht die Ordnung, Leistung und Motivation des Teams aufrecht zu halten.

Kategorisierung von Netzwerkverbindungen Umsetzung (Ziel H13, Seiler)
Das Konzept zur Kategorisierung von Netzwerkverbindungen ist praktisch umgesetzt.

Mithilfe des durch das Partnerteam Cerebrum erstellten Konzepts zur Kategorisierung werden Netzwerkverbindungen in Kategorien eingeteilt (z.B.: Remote-Zugriff, Mail-Verkehr, Streaming, ...). Diese werden mittels Skripts automatisch ermittelt und der Datenstruktur übergeben.

1.1.2 Optionale Ziele

Optionale Ziele sind für den Erfolg der Diplomarbeit nicht notwendig. Sie stellen einen gewissen Bonus dar und werden nur bearbeitet, wenn der Aufwand nicht zu groß ist und noch genügend Zeit vorhanden ist.

Ausweitung der SNMP-Erfassung (Ziel O1, Kaiser)
Das Ziel „Erfassung der quantitativen Datenmenge eines Klassenraums (Ziel H5, Kaiser)“ ist auf weitere Klassen und falls möglich auf das ganze Schulgebäude ausgeweitet.

Der SNMP-Manager fragt periodisch die mit den Klassen verbundenen Switches ab. Die Switches sind mit entsprechendem Community-String und Passwort konfiguriert. Nach Abfrage durch den SNMP-Manager schicken die Switches dem SNMP-Manager die abgefragten Daten. Mittels eines Skripts werden die Daten in einer passenden Struktur gespeichert und der Datenstruktur übergeben.

Ausweitung der NetFlow-Erfassung (Ziel O2, Seiler)
Das Ziel „Erfassung aktiver Netzwerkverbindungen eines Klassenraums (Ziel H10, Seiler)“ ist auf weitere Klassen und falls möglich auf das ganze Schulgebäude ausgeweitet.

Alle ein- und ausgehenden TCP-, UDP- und ICMP-Verbindungen der Klassen werden mittels dem Traffic-Analyzer NetFlow erfasst und in dem einzurichtenden NetFlow-Collector gespeichert. Die gesammelten Daten werden aus dem Netflow-Collector mittels eines Skripts der Datenstruktur übergeben. Eventuell muss ein Packet Inspector (z.B. Wireshark) aufgrund mangelnder Performance der Netzwerkgeräte verwendet werden.

Echtzeitaktualisierung Tagesstatistiken (Ziel O3, Krenn)
Die Tagesstatistiken einzelner Teilbereiche (SNMP, Netzwerkverbindungen, Angriffe & Programm- und Firmenzuordnung) sind mit periodischen Updates aktualisiert.
Bei jedem neuem Update einer der Teilbereiche werden die aktuellen Statistiken abgefragt. Der neue zu speichernde Datensatz wird miteinberechnet und anschließend in der Datenstruktur gespeichert.

Ausweitung der Portscan-Erfassung (Ziel O4, Kaiser)
Das Ziel „Erkennung von Portscans (Ziel H6, Kaiser)“ ist auf weitere Switches im Raum 072 und 079 und zusätzliche Firewalls ausgeweitet.

Die Mustererkennung für Portscans wird auch auf weiteren Geräten eingesetzt. Die somit gewonnen Daten werden ebenfalls der Datenstruktur übermittelt.

1.1.3 Nicht-Ziele

Um den Umfang der Diplomarbeit eingrenzen zu können, werden Nicht-Ziele formuliert. Diese zeigen auf, was nicht geliefert wird und schließen aus, was sich der Auftraggeber im Zuge der Diplomarbeit erwarten könnte.

Visualisierung der Daten
Die gesammelten Daten der Diplomarbeiten Nexus und Cerebrum sind auf einer Website grafisch visualisiert.

Konzept zur Kategorisierung von Netzwerkverbindungen
Anhand wesentlicher Eigenschaften (TCP/UDP und Portnummer) einer Netzwerkverbindung ist eine Liste aus groben Kategorien (z.B.: Video, Stream, Websiteaufruf) erstellt.

Konfigurieren der Packet Inspection Infrastruktur
Auf den, für die Packet Inspection benötigten, Netzwerkgeräten der Schule wird jeweils ein Mirrorport konfiguriert. Bei jedem Mirrorport wird bereits ein Filter angewendet, um den Netzwerkverkehr auf das Notwendigste zu beschränken. Zusätzlich ist eine Linux-VM (Sniffer-VM) aufgesetzt, die den gefilterten Traffic des Mirrorports empfängt.

1.2 Ergebnisse

Beim Endprodukt dieser Diplomarbeit handelt es sich um Skripte zur Datenanalyse, Skripten zur Befüllung der Datenbank sowie um eine API, welche die Datenbereitstellung für „Oculus“ ermöglicht. Zusammen ergibt dies eine einfache Möglichkeit, einen guten Überblick über das eigene Netzwerk zu erhalten. Sämtliche Ziele dieser Diplomarbeit wurden erfüllt und vom Auftraggeber abgenommen.

Die Ergebnisse der Diplomarbeit sind somit:

- Ein Skript zur Erfassung der Datenmenge (Ziel H5 und Ziel O1, siehe Kapitel 4.5)

- Ein Skript zur Erkennung von Portscans (Ziel H7 und Ziel O4, siehe Kapitel 5.2)
- Ein Skript zur Erfassung und Kategorisierung von Netzwerkverbindungen (Ziel H10, Ziel H13 und Ziel O2, siehe Kapitel 6.2.2 und Kapitel 7.1)
- Eine Datenbankstruktur für eine MongoDB-Datenbank (Ziel H1 und Ziel H3, siehe Kapitel 8.1.3)
- Ein Skript zum Befüllen der Echtdaten-Datenbank (Ziel H2, siehe Kapitel 8.1.4)
- Zwei Star-Schemen für die Statistik-Datenbank (Ziel H1 und Ziel H3, siehe Kapitel 8.2.3.1)
- Ein Skript für die Erstellung der Statistiken (Ziel H4, siehe Kapitel 8.2.4)
- Automatisierte Jobs für das Managen der anfallenden Dateien (Ziel H11, siehe Kapitel 9)
- Ein API-Konzept für den Gebrauch der erstellten API (Ziel H8, siehe Kapitel 10.1)
- Eine API für die Datenbereitstellung (Ziel H9, siehe Kapitel 10.2)

2 Specto

Die Diplomarbeit „Specto“ ermöglicht es Netzwerkadministratoren einen besseren Überblick über deren Netzwerke zu behalten. Dabei ist es für diese nicht immer möglich, auf kostenpflichtige Tools zurückzugreifen. Daher möchte „Specto“ im Rahmen dieser Diplomarbeit der HTL Rennweg eine Alternative bieten: Ohne zusätzlichen Kosten wird das Monitoring des Netzwerkes möglich gemacht.

Im Rahmen dieser Arbeit befasst sich „Specto“ mit der Erfassung des Datenverkehrs einer gesamten Schule – der HTL Rennweg. Die gesammelten Daten werden zentral gespeichert, zu Statistiken weiterverarbeitet und via API für die weitere externe Nutzung bereitgestellt. Die API ermöglicht es anschließend die Daten auf einer Website anzuzeigen, welche schließlich einen Gesamtüberblick des Netzwerkes bietet. Dabei musste stets beachtet werden, dass die Sicherheit und die Performance des Netzwerkes nicht unter den Eingriffen leiden.

Dem dabei aufkommenden Aufwand entsprechend unterteilt sich „Specto“ in die folgenden drei Diplomarbeiten, welche gemeinsam an dem großen Ziel arbeiten:

- Cerebrum - Traffic-Analyse
- Nexus - Traffic-Analyse, Datenbereitstellung
- Oculus - Visualisierung

2.1 Big Picture

Da die Diplomarbeit „Specto“ für Außenstehende ziemlich komplex wirkt, kann diese anhand eines „Big Pictures“ besser erklärt werden. Der logische Aufbau ist in Abbildung 1 grafisch dargestellt.

Wie aus der Grafik hervorgehend stellt das *Corporate Network* (CN) eine Verbindung zwischen den von „Specto“ benötigten und bearbeiteten Zonen dar. Der Sniffer-Host ermöglicht es „Specto“ sämtliche ins Internet gehend und aus dem Internet kommende Daten der HTL Rennweg zu erfassen. Dies ermöglicht ein *Mirror-Port* am Uplink der Schule, der alle Pakete an den Sniffer-Host weiterleitet.

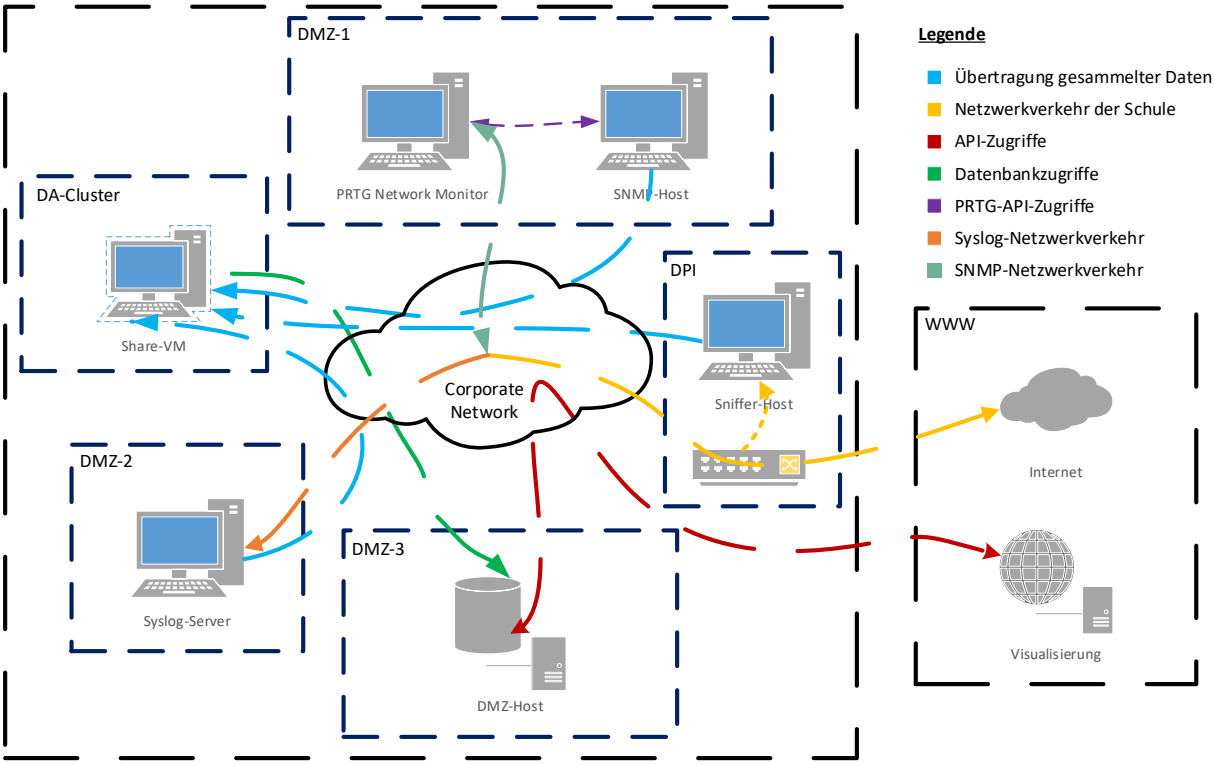


Abbildung 1. Gesamtüberblick über „Specto“

Die Zonen DMZ-1 und DMZ-2 dienen zur Beschaffung von SNMP- beziehungsweise Syslog-Daten. Der DA-Cluster bietet „Specto“ eine Menge an Ressourcen für virtuelle Maschinen (VMs), wobei die Share-VM den zentralen Verwaltungspunkt darstellt. Der logische Aufbau des DA-Clusters kann aus Abbildung 2 entnommen werden. Der DMZ-Host, in DMZ-3 terminierend, hostet die beiden Datenbanken sowie die API. Dahingehend ist dieser für externe Nutzung verfügbar. Die Visualisierung findet extern von Oculus statt und greift auf die von „Nexus“ bereitgestellte API zu.

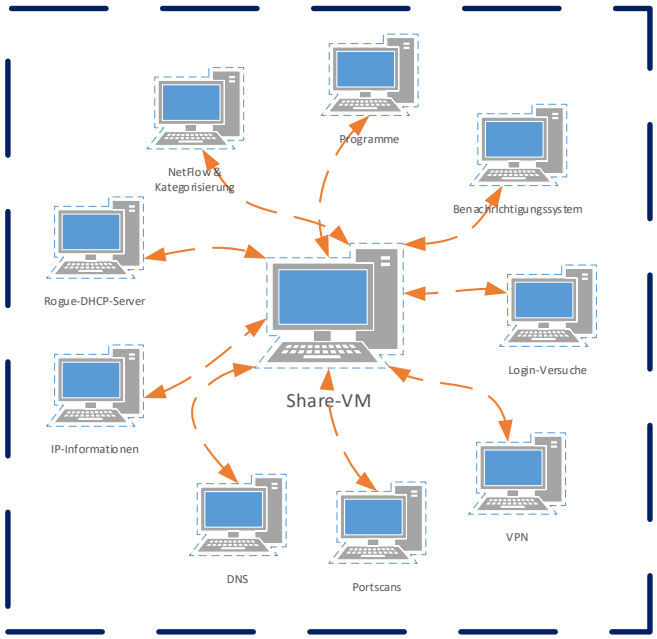


Abbildung 2. Überblick der Bereiche im DA-Cluster

2.2 Nexus

Die Diplomarbeit „Nexus“ ist für die Erfassung der Daten, für die zentralisierte Speicherung der gewonnenen Daten, für die Erstellung sowie Speicherung von Statistiken und für die Datenbereitstellung via API verantwortlich. „Nexus“ ist für die aus Abbildung 1 hervorgehende Zonen DMZ-1 und DMZ-3 zuständig. Weiters werden folgende Bereiche des DA-Clusters aus Abbildung 2 in Form von VMs von der Diplomarbeit „Nexus“ verwaltet:

- Portscans
- NetFlow & Kategorisierung
- Share-VM

2.2.1 Portscans

Die für Portscans zuständige VM prüft PCAP-Dateien auf ausgeführte Portscans. Informationen zu den aufgedeckten Portscans werden anschließend an die Share-VM übertragen.

2.2.2 NetFlow & Kategorisierung

Für die Bearbeitung der Bereiche NetFlow und Kategorisierung steht eine VM zur Verfügung. Diese liefert die aus PCAP-Dateien gewonnenen Informationen und in Kategorien eingeteilten Netzwerkverbindungen an die Share-VM.

2.2.3 Share-VM

Die Share-VM ist das zentrale Element des DA-Clusters. Diese VM stellt einen zentralen Sammelpunkt für die gesammelten Daten bereit. Auf diesen pushen alle anderen VMs im DA-Cluster sowie die Maschinen aus den anderen Zonen (SNMP-Host, Syslog-Sever und Sniffer-Host) die von ihnen gesammelten Daten. Die Share-VM ist zusätzlich für das Übertragen der gesammelten Datensätze an die auf dem DMZ-Host laufende Echtdaten-Datenbank zuständig. Weiters erstellt die Share-VM täglich Statistiken, welche anschließend in der Statistik-Datenbank auf dem DMZ-Host gespeichert werden.

3 Projektmanagement

Um die Diplomarbeit „Specto“, bestehend aus deren Unterprojekten „Cerebrum“, „Nexus“ und „Oculus“, zu managen, musste eine dem Umfang der Diplomarbeit gerechte Projektmanagementmethode ausgewählt werden. Hierbei entschieden sich die Teams Cerebrum und Nexus für einen Hybriden aus den Methoden *Wasserfall* und *Scrum*. Dadurch erhielten die beiden Teams die Vorzüge aus beiden Projektmanagementmethoden.

Durch *Wasserfall* wurden die Ziele beider Teams zu Beginn der Diplomarbeit genau definiert. Zusätzlich wurden die Umfelder und die Risiken abgeschätzt und beachtet. Auch die Projektstruktur bestehend aus einem *Projektleiter*, einem *-stellvertreter* und einem *-mitglied* wurde aus dieser Methode entnommen.

Durch *Scrum* behielten beide Teams Flexibilität und konnten leichter Änderungen an Aufgaben durchführen. Außerdem wurden alle Betreuer regelmäßig über den aktuellen Status der Diplomarbeiten informiert.

Mit dem Verfassen des Diplomarbeitsantrags wurden die Ziele, genau wie das Umfeld und die Risiken, der Diplomarbeit Nexus definiert. Daraufhin wurden diese Ziele als Grundlage für allgemeine Aufgaben aller Projektmitglieder benutzt.

Ab Mitte November empfand das Team die bisherige Leistung als nicht befriedigend und die zuvor definierten Ziele wurden fortan in wöchentliche Aufgaben heruntergebrochen. Dadurch waren die Berichterstattung und die Einsicht auf den Fortschritt einfacher.

3.1 Berichterstattung

Die Betreuer mussten regelmäßig über den aktuellen Stand der Diplomarbeit Nexus informiert werden. Hierzu wurde versucht jede zweite Woche ein gemeinsames Meeting mit dem Partnerteam Cerebrum und allen Betreuern zu führen. In den Wochen, in denen kein Statusmeeting stattfand, wurden Management-Summaries verfasst und an alle Betreuer versendet.

3.2 Product-Backlog

Die heruntergebrochenen Aufgaben der Hauptziele wurden nummeriert und im Product-Backlog festgehalten. Für jede Aufgabe schätzten die Teammitglieder ihren Zeitaufwand. Nach Abschluss jeder Aufgabe, wurde die tatsächlich verwendete Zeit festgehalten, damit spätere wöchentliche Aufgaben zeitlich genauer geschätzt werden konnten.

Nummer	Als ...	kann ich ...	um ... zu können.	benötigt	Geplante Stunden	Absolvierte Stunden	Erledigt
SEI_1	Collector-VM	automatisch PCAP-Dateien in einem Ordner auswerten	diese verarbeiten		0,5	0,5	ja
SEI_2	Collector-VM	automatisch fertige Textdateien an die Datenbank-VM schicken	die Daten weiter verarbeiten		1	0,5	ja

Abbildung 3. Auszug aus dem Product-Backlog

4 Erfassung der Datenmenge pro Klassenraum

Die Mitverfolgung des Datenverbrauchs in einem Netzwerk bietet einem Administrator einen guten Überblick darüber, wie viele Daten schulweit übertragen werden. Damit dies auch im Fall der vorliegenden Diplomarbeit möglich ist, wird SNMP verwendet. Hierzu wird, wie im Folgenden genauer erläutert, auf den Switches in der Schule ein Community-String eingerichtet, wodurch die Datenmenge der Klassen über eine Management-Software abgefragt werden kann.

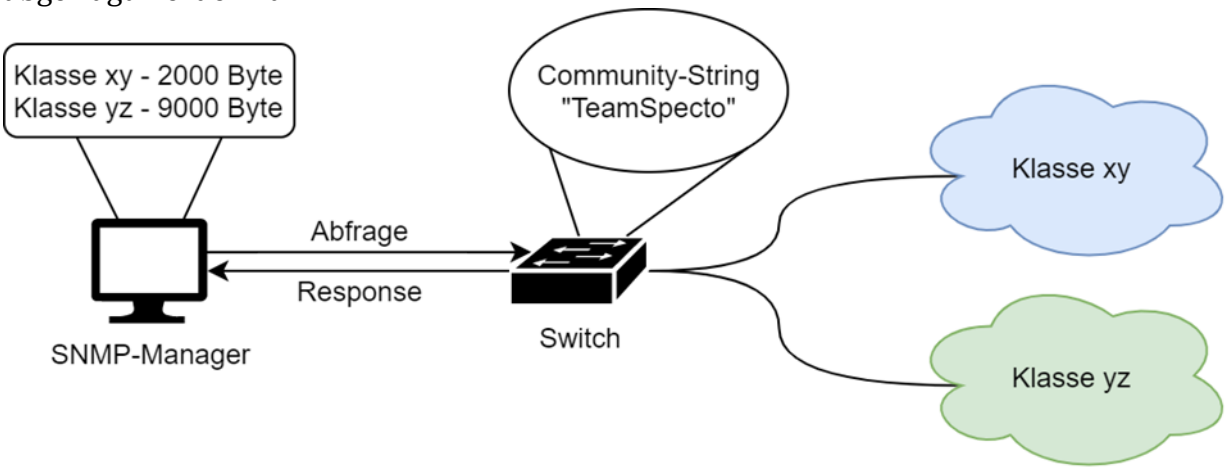


Abbildung 4. SNMP-Veranschaulichung

4.1 SNMP

Das Simple Network Management Protocol ist eines der weitverbreitetsten Protokolle zur Netzwerküberwachung beziehungsweise Verwaltung. Es wird trotz der verschiedenen Hersteller einzelner Netzwerkkomponenten und der unterschiedlichen, darauf laufenden Software eine einheitliche Kommunikationsbasis im Netzwerk geschaffen. Von SNMP [1] werden drei verschiedene Versionen unterstützt, die sich besonders hinsichtlich des Sicherheitsfaktors der übermittelnden Daten unterscheiden:

4.1.1 SNMPv1

Die erste Version von SNMP wurde 1988 definiert. Für die Implementierung wird die Einrichtung eines Community-Strings, welcher den Zugriff vom SNMP-Manager auf die SNMP-Daten eines SNMP-Agents ermöglicht, benötigt. Dieser wird als ein textbasiertes Passwort

[1] Simple Network Management Protocol

angelegt und wird in Klartext übertragen. Da es sich hierbei um die einzige Sicherheitsvorkehrung handelt, gilt diese Version somit als nicht sicher. (vgl. Paessler o. J.; DPS Telecom o. J.)

4.1.2 SNMPv2

Die zweite Version kann in drei Unterversionen unterschieden werden: SNMPv2c, SNMPv2u und SNMPv2p.

- SNMPv2c verwendet das Sicherheitsmodel von SNMPv1, das auf einem Community-String basiert. Im Unterschied zur ersten Version werden hier auch 64-Bit-Prozessorarchitekturen unterstützt, wodurch eine bessere Performance ermöglicht wird. Darüber hinaus ist diese Variante am weitesten verbreitet und wird häufig auch einfach als SNMPv2 adressiert. (vgl. Wikimedia Foundation 2010; vgl. Paessler o. J.)
- SNMPv2u versucht, mehr Sicherheit zu bieten. Hierzu werden zusätzlich Benutzernamen zur Authentifizierung genutzt. Diese Variante kommt heute nicht mehr zum Einsatz. (vgl. Wikimedia Foundation 2010)
- SNMPv2p überträgt den Community-String verschlüsselt. Jedoch wird diese Version als zu komplex angesehen und wurde daher nicht standardmäßig übernommen. (vgl. Wikimedia Foundation 2010)

Von den drei Unterversionen hat sich nur SNMPv2c durchgesetzt. Es wird zwar immer noch dasselbe Sicherheitsmodell wie in Version 1 verwendet, dafür gibt es in SNMPv2 eine optimierte Fehlerbehandlung. Weiters sind die beiden neuen Nachrichtentypen GETBULK und INFORM hinzugekommen. Nähere Informationen zu den Nachrichtentypen sind im Kapitel „Nachrichtentypen“ zu finden. (vgl. IONOS 2018)

4.1.3 SNMPv3

Die dritte Version von SNMP hebt sich besonders hinsichtlich der Sicherheitsfunktionen von den vorhergehenden Versionen ab. So können die übertragenen Daten nun durch Authentifizierung sowie Verschlüsselung vor unbefugten Zugriffen geschützt werden. Dementsprechend steigt mit dieser Version auch der Leistungsverbrauch der Netzwerkgeräte im Vergleich zu den anderen Versionen an. Die neuen Sicherheitsfunktionen sind in drei verschiedenen Sicherheitsstufen etabliert: **NoAuthNoPriv**, **AuthNoPriv** und **AuthPriv**. (vgl. Paessler o. J.)

- **NoAuthNoPriv** stellt die niedrigste Sicherheitsstufe dar. Hier gibt es keine Authentifizierung und auch die Daten werden nicht verschlüsselt. (vgl. Paessler o. J.)
- **AuthNoPriv** ist die mittlere Sicherheitsstufe. Eine Authentifizierung ist notwendig, um Daten bearbeiten oder schicken zu können. Die Daten selbst werden nach wie vor nicht verschlüsselt übertragen. (vgl. Paessler o. J.)
- **AuthPriv** bietet die höchste Sicherheit. Neben der Authentifizierung, um auf Daten zugreifen zu können, wird nun auch die Übertragung verschlüsselt. (vgl. Paessler o. J.)

Für die Kommunikation werden zwei Komponenten benötigt: der SNMP-Manager und SNMP-Agents. Zwischen den beiden besteht ein bidirektionaler Kommunikationskanal.

4.1.4 SNMP-Manager

Der SNMP-Manager ist eine spezielle Software, die für die Herstellung der Kommunikation mit den SNMP-Agents zuständig ist. Dadurch können auch mehrere Geräte gleichzeitig verwaltet werden, ohne dass ein Administrator sich mit jedem einzeln verbinden muss. Die Hauptfunktionen sind das Abfragen von Agents sowie das Abrufen von Antworten von Agents oder das Einrichten von Variablen auf Agents. In einem Netzwerk kann auch mehr als ein SNMP-Manager vorhanden sein. (vgl. Wikimedia Foundation 2010)

4.1.5 SNMP-Agent

Der SNMP-Agent ist eine Software, die auf den zu verwaltenden Geräten installiert ist. Durch sie wird laufend der Gerätestatus überwacht und die Informationen werden an den SNMP-Manager geschickt. Ist der Agent auf einem Gerät aktiviert, so wird von diesem aus die MIB (Management Information Base) des Geräts verwaltet und sichergestellt, dass der SNMP-Manager auf die entsprechenden Daten zugreifen kann. (vgl. Wikimedia Foundation 2010)

4.1.6 Management Information Base (MIB)

Wie im vorherigen Abschnitt erwähnt, wird für die Zugriffe auf die Daten – also für die Abfragen – die sogenannte Management Information Base (MIB) verwendet, die vom jeweiligen Agent des zu überwachenden Geräts verwaltet wird.

Die MIB enthält die überwachbaren Komponenten als sogenannte OIDs (Object Identifiers) in einem hierarchischen Format. Damit man von der Management Software aus auf die MIB zugreifen kann, muss man sich zunächst mittels des Community-Strings bei SNMPv1 und SNMPv2 beziehungsweise mit dem Usernamen bei SNMPv3 authentifizieren. Bei den Community-Strings kann in weiterer Folge zwischen Read-only und Read-write unterschieden werden. Dementsprechend kann man entweder nur Daten auslesen oder auch Daten verändern. Für einen Zugriff muss das zu überwachende Ziel in der MIB lokalisiert werden und entsprechend über die hierarchische Struktur des MIB-Baums (siehe Abbildung 5) eine Abfolge der einzelnen OIDs abgefragt werden. Die OIDs werden mittels Punkt aneinandergereiht. (vgl. Paessler o. J.)

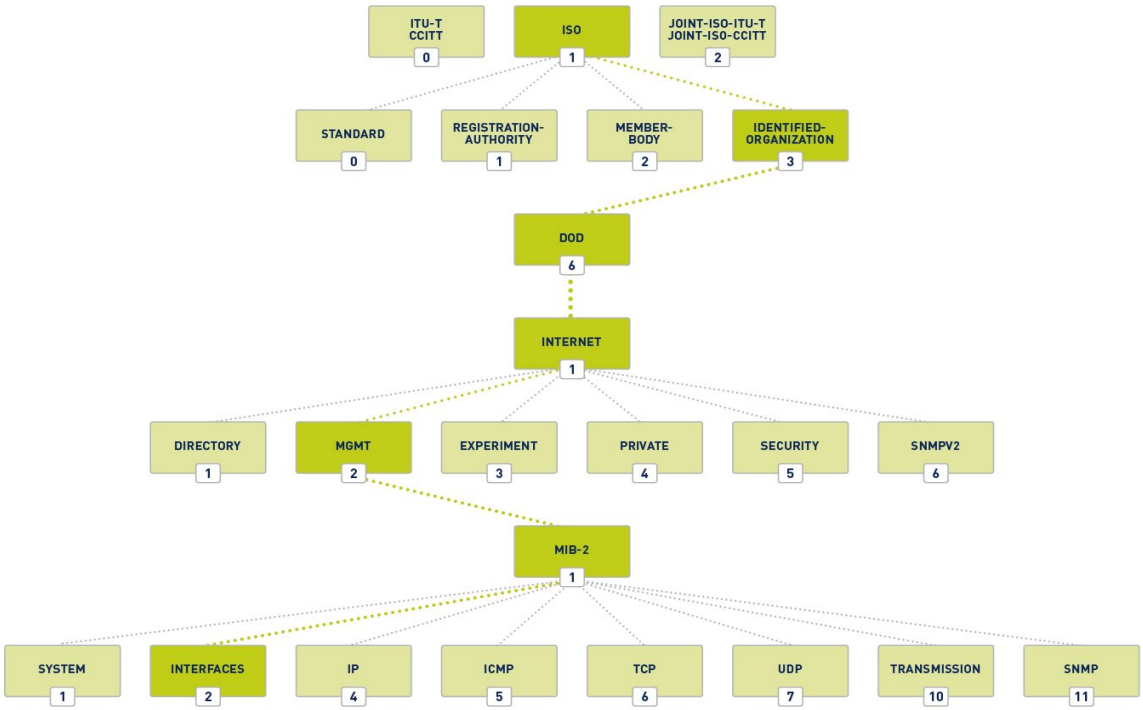


Abbildung 5. Beispiel einer MIB-Baum-Struktur.
Quelle: https://kb.paessler.com/en/topic/653-how-do-snmp-mibs-and-oids-work

4.1.7 SNMP-Nachrichten

Für die Kommunikation zwischen dem SNMP-Manager und den SNMP-Agents kommen verschiedene SNMP-Nachrichten zum Einsatz. Anhand derer können Informationen der MIB abgefragt oder auch verändert werden.

Get-Requests dienen zur Abfrage von Informationen. Sie sind in den meisten Fällen die Hauptmethode von SNMP-Managern, um Informationen von den Agents zu erhalten. Hierbei können ein oder mehrere spezielle Werte aus der MIB abgefragt werden. (vgl. IONOS 2018)

GetNext-Requests werden verwendet, um den Wert des nächsten OIDs im MIB-Baum abrufen zu können. (vgl. IONOS 2018)

GetBulk-Requests können verwendet werden, um ein Array von Informationen über eine Folge von GetNext-Requests anzufordern. Dadurch kann ein größeres Segment der MIB abgefragt werden. (vgl. IONOS 2018)

Set-Requests dienen dazu, dem Agent Anweisungen für Änderungen an den Einstellungen eines überwachten Geräts zu schicken. (vgl. IONOS 2018)

Get-Responses werden vom Agent geschickt, um Anforderungen zu bestätigen. Diese Nachricht wird als Antwort auf Get-, GetNext-, GetBulk- und Set-Requests geschickt. (vgl. IONOS 2018)

Traps dienen als Warnnachrichten, die den SNMP-Manager beispielsweise über den Ausfall einer zu überwachenden Komponente informieren. Sie werden daher ausgehend vom SNMP-Agent geschickt. (vgl. IONOS 2018)

Inform-Requests haben den gleichen Zweck wie Traps, sie werden jedoch vom SNMP-Manager bestätigt. (vgl. IONOS 2018)

Die Anfragen des SNMP-Managers erfolgen auf dem UDP Port 161 und die Nachrichten, die ausgehend vom SNMP-Agent geschickt werden, erfolgen auf dem UDP Port 162 (vgl. IONOS 2018).

4.2 In der Diplomarbeit eingesetzte Version von SNMP

Für die Implementierung von SNMP in der Schule kommen die Versionen SNMPv2c und SNMPv3 in Frage. SNMPv1 und SNMPv2c ähneln sich sehr, mit dem Unterschied, dass bei der 1. Version weniger Features unterstützt werden. Aufgrund dessen fiel diese Version bei der Auswahl von vornherein weg. Die Entscheidung fiel daher auf SNMPv2c anhand für diese Diplomarbeit relevanter Kriterien, die im Folgenden erläutert werden.

Eine optimale Funktionalität im Schulnetzwerk setzt eine SNMP-Version voraus, die sich einfach und ohne Auswirkungen im laufenden Produktivnetzwerk implementieren lässt. Besonders die Netzwerkauslastung sollte nicht aufgrund der Ergänzung von SNMP im Netzwerk verschlechtert werden. Dies ist besonders im Fall einer Schule, in der viel täglicher Netzwerkverkehr besteht, relevant. Zuletzt ist es ebenso wichtig, dass die Version mit der folglich verwendeten SNMP-Manager-Software kompatibel ist und dahingehend keine Probleme auftreten.

Aus den oben genannten Gründen fiel die Entscheidung auf SNMPv2c. In dieser Version wird für die Konfiguration lediglich ein Community-String benötigt, der im Weiteren nur eine Leseberechtigung auf die MIB benötigt. Dadurch ist zwar kaum Sicherheit vor Fremdzugriffen geboten, jedoch sollte dies wegen der durch SNMP gewonnen Daten kaum eine Rolle spielen, da wirklich nur die Datenmenge auf den Interfaces der Switches erfasst wird. Ebenso können Schüler und Schülerinnen ohnehin nicht auf den Datenverkehr zwischen den Switches und dem SNMP-Manager zugreifen, ohne sich illegalen Zugriff auf das Netzwerk zu verschaffen. Ein weiterer Grund für SNMPv2c ist, dass keine weitere Netzwerkauslastung entsteht. Bei SNMPv3 geht mehr Bandbreite und Leistung aufgrund der Verschlüsselung und der Authentifizierung verloren. Bezüglich der SNMP-Manager-Software sind beide kompatibel, jedoch wird bei der ausgewählten Software, um die es im folgenden Abschnitt geht, empfohlen, nicht zu viele SNMPv3 Sensoren aus Performancegründen zu verwenden.

4.3 Ausgewählte Software

Als Kriterien für die eingesetzte SNMP-Manager-Software sind besonders die Unterstützung der gewählten SNMP-Version und die Möglichkeit der Nutzung ohne Trial-Version wichtig. Es sollte auch keine Einschränkungen geben, wenn die Gratisversion genutzt wird. Ebenso muss die Implementierung dieser Software auf einem Host im Schulnetzwerk auch für den Schuladministrator vertretbar sein.

„PRTG Network Monitor“ wurde als SNMP-Manager-Software gewählt. Der ausschlaggebende Grund hierfür war, dass die Software bereits produktiv im Schulnetzwerk eingesetzt wird. So würde die Software zwar auch als Gratisversion zur Verfügung stehen und funktionieren, jedoch hat die Schule bereits eine laufende Lizenz, die mitverwendet werden kann. Des Weiteren werden alle SNMP-Versionen unterstützt und es kann auf einfache Weise die gewünschte SNMP-Abfrage der Datenmenge auf den Switch-Interfaces durchgeführt werden. Weitere Informationen diesbezüglich sind im Kapitel „Implementierung in der Schule“ zu finden. Ein weiterer Vorteil dieser Software ist, dass es auch ein User-Management gibt und so für die Diplomarbeit ein eigener Nutzer angelegt werden kann, wodurch die schulinternen Daten von denen, die „Specto“ benötigt, getrennt werden können.

4.4 Implementierung von SNMP in der Schule

Nachdem die Wahl der SNMP-Version und der SNMP-Manager-Software getroffen worden war, fehlte noch die Implementierung von SNMP im Schulnetzwerk. Diese wurde zunächst in einer Testumgebung durchgeführt und getestet. Daraus ist eine entsprechende Vorgangsdokumentation entstanden, die in weiterer Folge für die tatsächliche Implementierung genutzt wurde, um Fehler zu vermeiden.

4.4.1 Ablauf

Für die Implementierung sind drei wesentliche Komponenten notwendig:

1. Switches als SNMP-Agents
2. ein Gerät als SNMP-Manager
3. ein Skript zur „tatsächlichen“ Informationsgewinnung

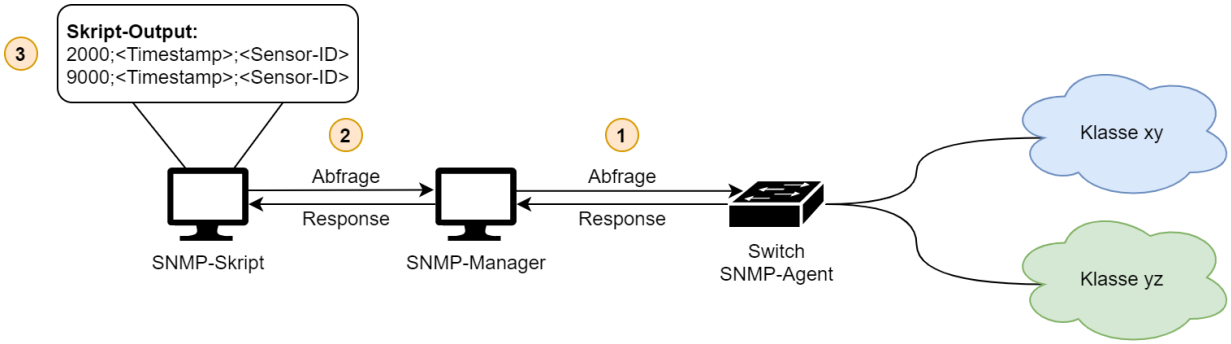


Abbildung 6. SNMP-Ablauf

Auf den Switches, die Interfaces zu den Klassenräumen haben, wird ein SNMP-Community-String eingerichtet. Dieser benötigt nur Read-only Rechte, da keine Änderungen vorgenommen werden sollen, sondern nur der Interface-Datenverkehr aus der MIB ausgelesen werden soll. Hierfür wird in der SNMP-Manager-Software der vorkonfigurierte Sensor **`SNMP Datenverkehr`** eingerichtet. Des Weiteren gibt es einen Host, der sich in einer DMZ (=Demilitarisierte Zone) befindet. Auf diesem läuft die SNMP-Manager-Software PRTG Networking Monitor. Damit dieser nun die Daten via SNMP erhält, müssen die Switches den Host auch erreichen können.

Letztendlich wird ein selbstgeschriebenes Python-Skript verwendet, das sich die von „Specto“ benötigten Informationen – die Datenmenge, den Zeitstempel und die Sensor-ID – holt. Dieses Skript muss ebenfalls auf einem Host in der DMZ laufen. Der PRTG Networking Monitor selbst würde zwar bereits eine grafische Darstellung der benötigten Informationen bieten, jedoch sind an dieser Stelle nur die vom Monitor gewonnenen Rohdaten für die Diplomarbeit relevant. Es wird von PRTG aber eine eigene interne API geboten, über die mittels des Python-Skripts die erforderlichen Informationen abgefragt werden können. Das Skript bringt die benötigten Rohdaten in das richtige Format und stellt sie als Textdatei (siehe Listing 9. Inhalt einer SNMP-Datei) bereit.

Damit sind die Datenmengen der einzelnen Klassenräume erfasst. Die fertige Textdatei wird aus datenschutzrechtlichen Gründen erst 48 Stunden nach der Erstellung mittels SMB zur Echtdaten-Datenbank gepusht. Nähere Information zur Weiterleitung der Daten sind im Kapitel „Sniffer-VM“ zu finden. Dort wird sie gespeichert und die Daten können vom Partnerteam Oculus grafisch in die Darstellung der Schule eingebunden werden.

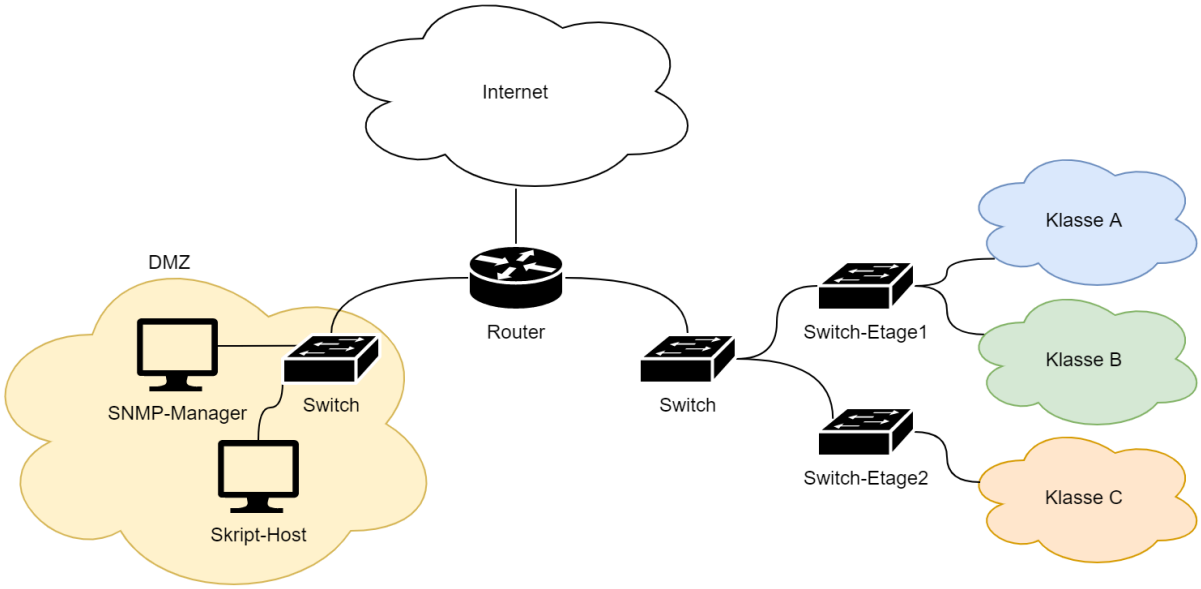


Abbildung 7. Vereinfachte SNMP-Topologie

4.4.2 Umsetzung der Implementierung von SNMP

Folglich müssen nun die benötigten Komponenten eingerichtet und konfiguriert werden. Wie bereits erwähnt, werden hierzu die gewonnenen Erkenntnisse aus der Testumgebung genutzt.

4.4.2.1 SNMP-Agents

Im Rahmen der Diplomarbeit muss auf jenen Switches, welche ein Interface zu einem zu überwachenden Klassenraum besitzen, SNMP konfiguriert werden. Das erfordert einerseits die Konfiguration einer Management IP-Adresse sowie die Angabe eines Default-Gateways. Mit folgenden Befehlen ist dies umsetzbar:

```
int vlan <Management-VLAN-ID>
ip address <IP-Adresse im Management-Netz> <SNM des Management-Netz>
no shut
exit
ip default-gateway <IP-Adresse des Routers im Management-Netz>
```

Listing 1. Switch-CLI: Management-VLAN Konfiguration

Zuletzt wird der SNMP-Community String **`TeamSpecto`** mit Read-only Rechten angelegt und die IP-Adresse des SNMP-Manager-Hosts angegeben. Bewerkstelligt wird dies auf den entsprechenden Switches wie folgt:

```
snmp-server community TeamSpecto ro
snmp-server host <IP-Adresse des SNMP-Managers> version 2c TeamSpecto
```

Listing 2. Switch-CLI: SNMP Konfiguration

4.4.2.2 SNMP-Manager

Wie im Kapitel „Ausgewählte Software“ erwähnt, wird der bereits im Schulnetzwerk eingesetzte PRTG Networking Monitor als SNMP-Manager verwendet. Dieser wird entsprechend für „Specto“ eingerichtet. Hierzu meldet man sich zuerst an und wechselt über die Menü-Leiste in den Bereich „Geräte“.

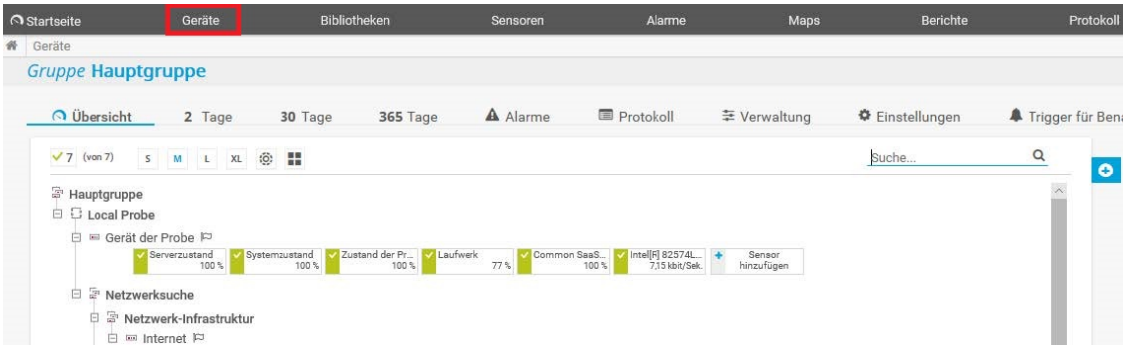


Abbildung 8. Geräte-Tab von PRTG Network Monitor

Sofern es den Switch nicht bereits als Gerät gibt, wird er zunächst hinzugefügt. Für eine bessere Übersicht kann davor noch über das „+“ rechts eine Gruppe für alle von „Specto“ benötigten Geräte erstellt werden.

Abbildung 9. Gruppe hinzufügen in PRTG Network Monitor

Die Gruppe benötigt lediglich einen Namen und ist damit fertig konfiguriert. Unterhalb der Gruppe befindet sich das Feld „Gerät hinzufügen“, über welches die benötigten Switches hinzugefügt werden. In dem darauffolgenden Fenster wird ein Name für das Gerät und die zugehörige Management-IP-Adresse des Switches angegeben.

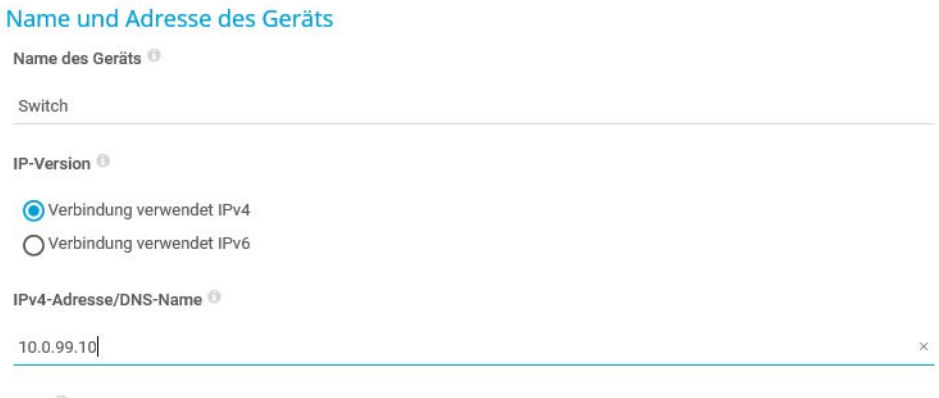


Abbildung 10. Gerät einrichten in PRTG Network Monitor

Ebenso wird weiter unten in den Einstellungen der Regler für das Übernehmen der Zugangsdaten für SNMP-Systeme deaktiviert und der am Switch spezifizierte Community-String wird angegeben.

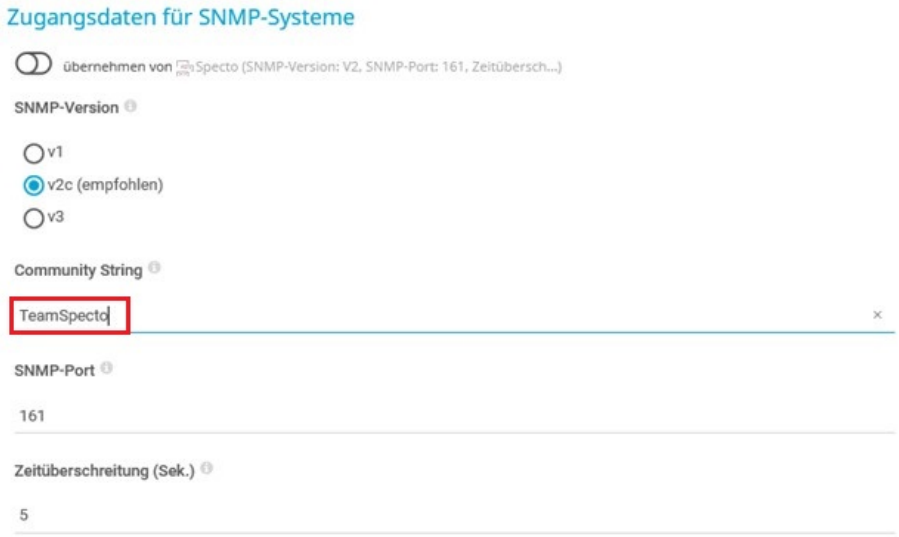


Abbildung 11. SNMP-Konfiguration eines Geräts in PRTG Network Monitor

Damit ist ein Gerät fertig hinzugefügt. Die Schritte für das Hinzufügen eines Gerätes werden für jeden Switch, über den Daten abgefragt werden sollen, wiederholt, sofern er nicht bereits im PRTG-Monitor vorhanden ist. Anschließend werden die Sensoren für den Datenverkehr auf den Geräten hinzugefügt. Hierfür gibt es unter dem jeweiligen Gerät das Feld „Sensor hinzufügen“. Im daraufhin erscheinenden Fenster wird bei der Frage „Was soll gemonitort werden?“ der Punkt „Bandbreite/Datenverkehr“ ausgewählt. Daraufhin wird bereits der benötigte Sensortyp „SNMP Datenverkehr“, der weiter unten erscheint, ausgewählt.

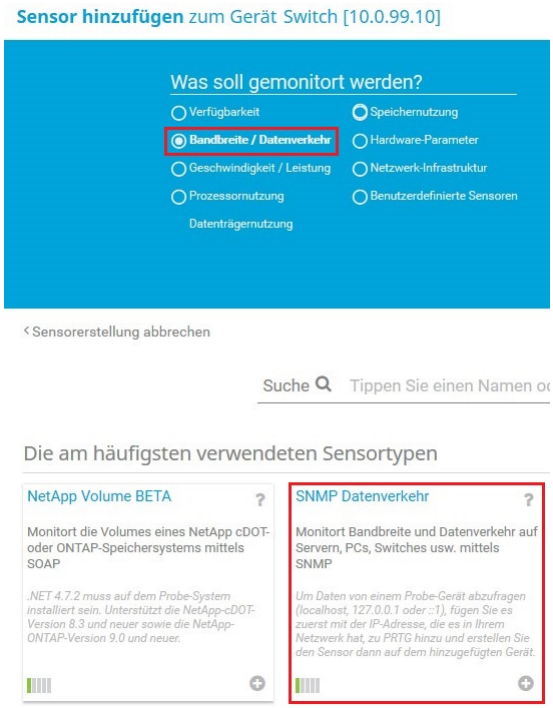


Abbildung 12. Sensor hinzufügen in PRTG Network Monitor

Zuletzt werden all jene Interfaces ausgewählt, die mit den zu überwachenden Klassenräumen verbunden sind. Die anderen Einstellungen werden bei der Default-Auswahl belassen.

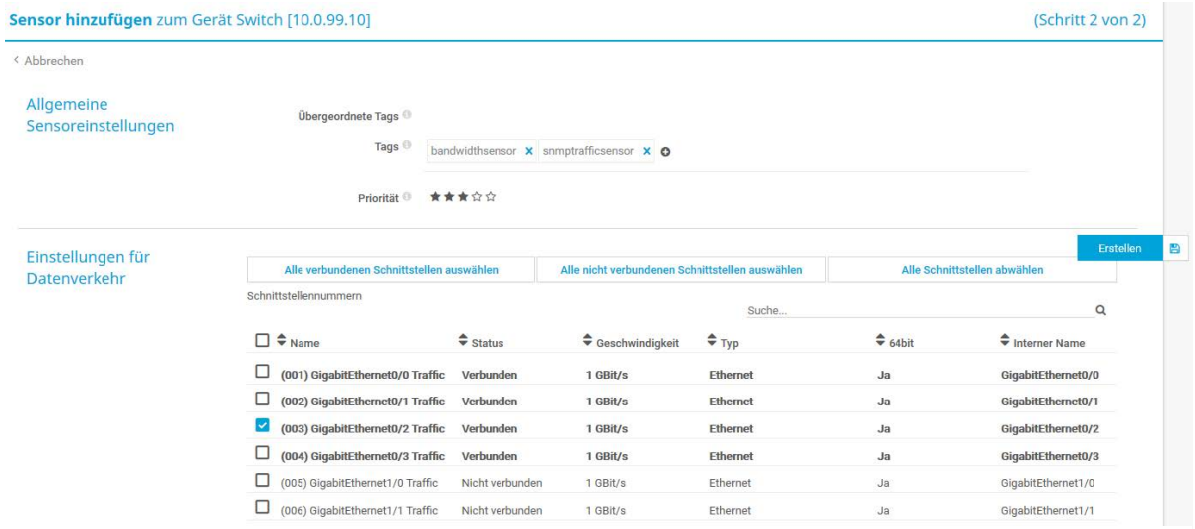


Abbildung 13. Interfaces auswählen in PRTG Network Monitor

Nachdem die Sensoren für alle Switches eingerichtet sind, ist die Konfiguration des SNMP-Managers abgeschlossen und er kann über das Skript abgefragt werden.

4.4.2.3 Skript-Host

Um die Daten vom PRTG Networking Monitor abzufragen, wird ein Python-Skript verwendet. Dieses wird automatisch auf einem Linux-Host ausgeführt. Hierfür muss aber zunächst der Host entsprechend eingerichtet werden.

Als Erstes wird hierzu im Terminal geprüft, ob und welche Version von Python installiert ist. Es sollte die Version 3.8 verwendet werden. Daraufhin wird die für das Skript benötigte Ordnerstruktur angelegt. Es werden die Ordner **`/home/specto/SNMP/Skript`** und **`/home/specto/SNMP/Logs`** benötigt. Im soeben erstellten Skript-Ordner wird eine .txt-Datei namens **`sensoren.txt`** erstellt. Die Verzeichnisstruktur sieht wie folgt aus:

```
/home/specto/
    ├── SNMP
    │   ├── Skript
    │   │   └── sensoren.txt
    │   └── Logs
```

Listing 3. Ausgabe der SNMP Verzeichnisstruktur im Linux-Terminal

Damit ist der Host selbst grundsätzlich fertig konfiguriert. Nun muss noch das Skript hinzugefügt und angepasst werden. Dazu wird im Terminal folgender Befehl ausgeführt:

```
nano /home/specto/SNMP/Skript/SnmpData.py
```

Listing 4. Linux-Terminal: SNMP Skript anpassen

Hier wird das Skript zur Abfrage des SNMP-Managers hineinkopiert, zuvor müssen aber noch folgende Änderungen vorgenommen werden. Das betreffende Skript wird im Kapitel

„Skript zur Datenmengenabfrage“ vorgestellt. Die zu ändernden Stellen sind im Skript wie folgt benannt:

- <IP-Adresse des PRTG Monitor Hosts>
- <PRTG-Username>
- <PRTG-Passwort>

Genauere Informationen zum Skript selbst sind im Kapitel „Skript zur Datenmengenabfrage“ zu finden. Nachdem das Skript nun angepasst ist, müssen die Sensor-IDs der zu überwachenden Interfaces unter **`/home/specto/SNMP/Skript/sensoren.txt`** angegeben werden. Die benötigten IDs sind im PRTG Networking Monitor unter „Geräte“ zu finden, wo der entsprechende Switch ausgewählt wird und die erstellten Sensoren nacheinander durchgegangen werden. Dort ist der Wert unter „ID“ hinter „#“ relevant.

Abbildung 14. Sensor-ID im PRTG Network Monitor nachschauen

Die einzelnen Sensor-IDs werden einfach mittels „ **;** “ in der .txt-Datei voneinander getrennt.

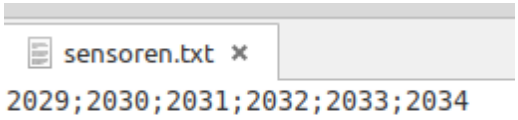


Abbildung 15. Beispiel einer sensoren.txt-Datei für das Skript zur SNMP-Manager-Abfrage

Damit die API-Abfrage im Skript durchgeführt werden kann, muss das Python-Modul „requests“ installiert werden. Hierfür muss zunächst auch noch „pip“ installiert werden. Diesbezüglich sind die folgenden Schritte im Terminal ausgeführt worden:

```
apt install python-pip
python -m pip install requests
```

Listing 5. Linux-Terminal: Python-Modul installieren

Zuletzt wird noch eingestellt, dass das Skript automatisch beim Start bzw. Neustart des Hosts ausgeführt wird. Dies erfolgt über einen crontab-Eintrag, indem in der crontab-Datei unter den Kommentaren der @reboot-Befehl mit dem Python-Aufruf des Skripts eingefügt wird.

```
@reboot python3 /home/specto/SNMP/Skript/SnmpData.py
```

Listing 6. Linux-Terminal: Crontab erstellen

Nach einem Neustart wird das Skript automatisch ausgeführt. Dies kann in der Datei **`/home/specto/SNMP/Logs/log.txt`** überprüft werden. Dort kommt bei jedem Start ein neuer Eintrag mit dem Text „Skript gestartet“ hinzu. Wartet man fünf Minuten, so wird auch schon der erste Datensatz erfasst und eine entsprechende Datei erscheint unter **`/home/specto/SNMP`**.

4.5 Skript zur Datenmengenabfrage

Damit die Datenmengen der Klassenräume im gewünschten Format zur Echtdaten-Datenbank (siehe „Echtdaten-Datenbank“) gelangen können, wird ein Python-Skript eingesetzt[2]. Wie im Kapitel „Ablauf“ erklärt, läuft das Skript hierzu auf einem Host in der DMZ und fragt den SNMP-Manager über die interne PRTG Networking Monitor API ab. Der genauere Ablauf des Skripts wird anhand des nachfolgenden Ablaufdiagramms veranschaulicht.

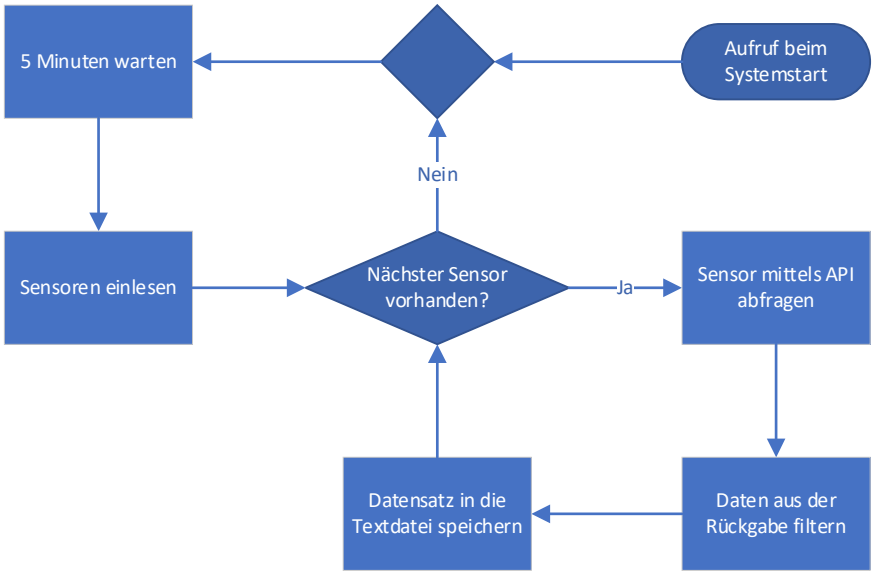


Abbildung 16. Ablauf: Datenmengen-Erfassung mittels SNMP

Für die Umsetzung im Skript werden zwei verschiedene Funktionen benötigt, auf die im Folgenden genauer eingegangen wird:

4.5.1 get_snmp_data()

Diese Funktion läuft in einer Dauerschleife, die immer fünf Minuten lang pausiert und anschließend erneut durchlaufen wird. In einem Durchlauf wird jeweils die erstellte **`sensoren.txt`**-Datei eingelesen und für jeden einzelnen darinstehenden Sensor ein API-Request auf den PRTG Networking Monitor durchgeführt. Dieser liefert daraufhin entsprechende Daten zurück. Ein solcher API-Request ist folgendermaßen aufgebaut:

[2] Das vollständige Skript ist als externe Datei „SnmpData.py“ auf der CD des Bibliotheksexemplars hinterlegt.

```
http://<IP-Adresse des PRTG Monitor
Hosts>/api/historicdata.xml?id=<Sensor-ID>&sdate=<%Y-%m-%d-%H-
%M>&edate=<%Y-%m-%d-%H-%M>&username=<PRTG-Username>&password=<PRTG-
Passwort>
```

Listing 7. Aufbau eines API-Requests zur Abfrage von Daten vom PRTG Network Monitor

Die in der oberen URL mit „<>“ gekennzeichneten Felder für die IP-Adresse, den Benutzernamen sowie das Passwort werden beim Einrichten, wie im Kapitel „Skript-Host“ erwähnt, entsprechend angepasst. Im Skript selbst wird eine solche Abfrage mit Hilfe von Variablen durchgeführt und wird wie folgt in einer **`for`**-Schleife über alle Sensoren der **`sensoren.txt`**-Datei iteriert:

```
for sensor in str(line.strip()).split(';'):
   url = url_start + sensor + '&sdate=' + start_time + '&edate=' +
         end_time + url_end
   req = requests.get(url, allow_redirects=True)
```

Listing 8. Python-Code: Durchführung des API-Requests und Speicherung der Ergebnisse

Nachdem geprüft worden ist, ob auch Daten zurückgekommen sind, werden über die Funktion **snmp_data_to_txt()** die relevanten Informationen herausgefiltert und zurückgeliefert. Anschließend wird die Rückgabe in die Ergebnisdatei geschrieben und ein entsprechender Log-Eintrag erstellt. Eine Ergebnisdatei kann anschließend folgendermaßen aussehen:

```
9212.3818;23.01.2021 16:55:00;2029
47923198.6248;23.01.2021 16:55:00;2030
403440.1756;23.01.2021 16:55:00;2031
39555015.1787;23.01.2021 16:55:00;2032
10160.0200;23.01.2021 16:55:00;2033
10260.0100;23.01.2021 16:55:00;2034
```

Listing 9. Inhalt einer SNMP-Datei

4.5.2 snmp_data_to_txt()

Diese Funktion wird von **get_snmp_data()** aufgerufen, um die von „Specto“ benötigten Daten zurückzuliefern. Hierzu werden der Funktion zum einen die Daten, die die API-Abfrage zurückgeliefert hat und zum anderen die aktuelle Sensor-ID übergeben. Die Daten selbst werden zunächst mittels **`if`**-Statement überprüft. Fängt der durch die API gewonnene textbasierte Output mit den Wörtern „Zu wenig“ an, was immer dann der Fall ist, wenn nicht genügend Daten aufgezeichnet werden konnten, wird **`0`** zurückgeliefert, damit die andere Funktion entsprechend darauf reagieren kann. Sind nun aber Daten vorhanden, so wird mittels *RegEx* der Zeitpunkt des Datensatzes und die Datenmenge herausgefiltert. Es handelt sich hierbei um recht einfache *RegEx*, die sich am Aufbau des Inhalts orientieren, da dieser immer gleich ist.

```
if data.startswith('Zu wenig'):
   return 0
else:
   timestamp = re.search("<[a-z]{8}>(.{19}) - ", data).group(1)
   traffic = re.search("<.{5}_.{57}>([0-9,.]*)", data).group(1)
```

Listing 10. Python-Code: im Skript angewandte RegEx für die Informationsgewinnung

Letztendlich liefert die Funktion die Daten, also die Datenmenge, den Zeitstempel und die Sensor-ID getrennt durch ein Semikolon als String zurück.

```
return traffic + ';' + timestamp + ';' + sensor + '\n'
```

Listing 11. Python-Code: Return-Statement mit den benötigten Daten

5 Portscans

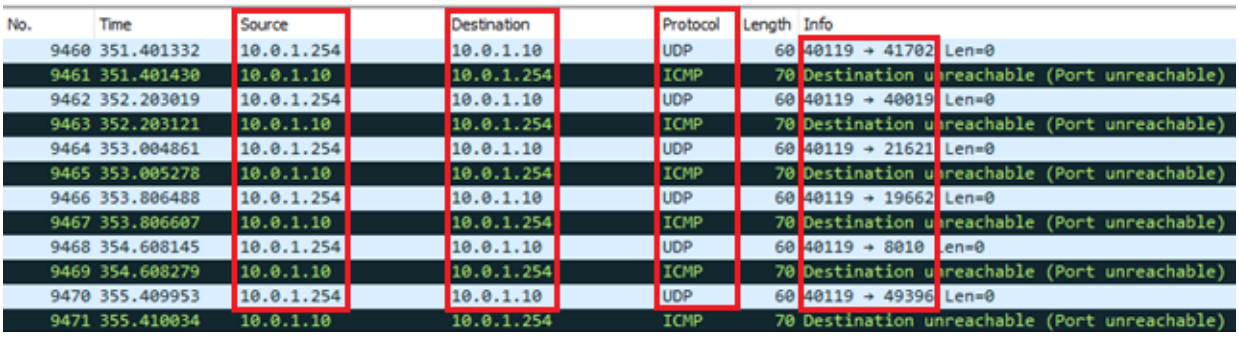
Ein Portscan ist häufig die erste Phase eines Angriffs auf ein Netzwerk. In dieser werden Informationen über das Netzwerk gesammelt. Dabei können unter anderem offene Ports, Betriebssysteminformationen sowie verwendete Programme von Netzwerkgeräten herausgefunden werden. Es findet in diesem Fall kein Angriff, wie man ihn sich normalerweise vorstellen würde, statt, da das Netzwerk nicht aktiv beeinträchtigt wird. Jedoch kann ein Angreifer auf diese Weise wichtige Informationen sammeln, die in weiterer Folge für einen für das Netzwerk schädlichen Angriff verwendet werden können. (vgl. Hagel o.J.; vgl. Winkler 2017)

Im Fall dieser Diplomarbeit sollen Portscans im Datenverkehr des Schulnetzwerks erkannt werden. Dadurch kann der Netzwerkadministrator besser abschätzen, wie häufig das Schulnetzwerk auf mögliche Sicherheitslücken geprüft wird und eventuell entsprechend mit Sicherheitsvorkehrungen reagieren. Hierzu werden die durch Team Cerebrum erzeugten *PCAP-Dateien* analysiert. Bei diesen handelt es sich um Dateien, die Informationen zu Netzwerkpaketen enthalten. Zur Erkennung von Portscans werden diese über ein selbstgeschriebenes Python-Skript mit Hilfe der Library „Scapy“ nach den im nächsten Abschnitt „Portscans in der Diplomarbeit“ definierten Kriterien durchsucht.

5.1 Portscans in der Diplomarbeit

Für die Erkennung von Portscans sind solche zunächst in einer Testumgebung mittels Wireshark betrachtet worden. Anders als zum Beispiel bei der VPN-Erkennung von Team Cerebrum, können bei der Beobachtung von Portscans in *Wireshark* aber keine eindeutigen Muster im *Payload* festgestellt werden. Jedoch können anhand der Source- und Destination-IP-Adressen oder auch anhand der Source- und Destination-Ports die Merkmale eines Portscans festgelegt werden.

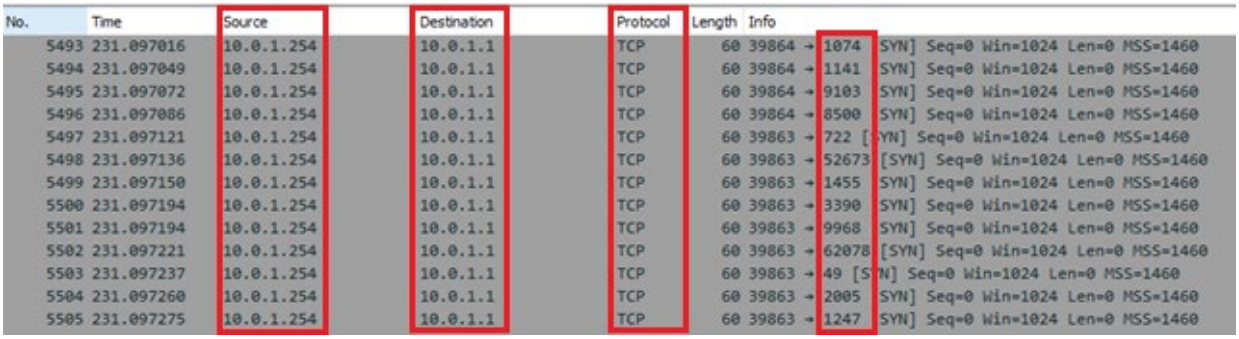
Bei einem UDP-Scan lässt sich in Wireshark erkennen, dass immer vom selben Host aus, also immer von der gleichen Source-IP-Adresse, Pakete an einen Ziel-Host mit der gleichen Destination-IP-Adresse geschickt werden. Weiters wird immer UDP als Protokoll verwendet und es werden verschiedene Ports auf dem Ziel-Host abgefragt.



No.	Time	Source	Destination	Protocol	Length	Info
9460	351.401332	10.0.1.254	10.0.1.10	UDP	60	40119 → 41702 Len=0
9461	351.401430	10.0.1.10	10.0.1.254	ICMP	70	Destination unreachable (Port unreachable)
9462	352.203019	10.0.1.254	10.0.1.10	UDP	60	40119 → 40019 Len=0
9463	352.203121	10.0.1.10	10.0.1.254	ICMP	70	Destination unreachable (Port unreachable)
9464	353.004861	10.0.1.254	10.0.1.10	UDP	60	40119 → 21621 Len=0
9465	353.005278	10.0.1.10	10.0.1.254	ICMP	70	Destination unreachable (Port unreachable)
9466	353.806488	10.0.1.254	10.0.1.10	UDP	60	40119 → 19662 Len=0
9467	353.806607	10.0.1.10	10.0.1.254	ICMP	70	Destination unreachable (Port unreachable)
9468	354.608145	10.0.1.254	10.0.1.10	UDP	60	40119 → 8010 Len=0
9469	354.608279	10.0.1.10	10.0.1.254	ICMP	70	Destination unreachable (Port unreachable)
9470	355.409953	10.0.1.254	10.0.1.10	UDP	60	40119 → 49396 Len=0
9471	355.410034	10.0.1.10	10.0.1.254	ICMP	70	Destination unreachable (Port unreachable)

Abbildung 17. UDP-Scan in Wireshark

Dasselbe trifft auch auf den TCP-Scan zu. Der einzige Unterschied liegt darin, dass TCP als Protokoll verwendet wird.



No.	Time	Source	Destination	Protocol	Length	Info
5493	231.097016	10.0.1.254	10.0.1.1	TCP	60	39864 → 1074 SYN] Seq=0 Win=1024 Len=0 MSS=1460
5494	231.097049	10.0.1.254	10.0.1.1	TCP	60	39864 → 1141 SYN] Seq=0 Win=1024 Len=0 MSS=1460
5495	231.097072	10.0.1.254	10.0.1.1	TCP	60	39864 → 9103 SYN] Seq=0 Win=1024 Len=0 MSS=1460
5496	231.097086	10.0.1.254	10.0.1.1	TCP	60	39864 → 8500 SYN] Seq=0 Win=1024 Len=0 MSS=1460
5497	231.097121	10.0.1.254	10.0.1.1	TCP	60	39863 → 722 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
5498	231.097136	10.0.1.254	10.0.1.1	TCP	60	39863 → 52673 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
5499	231.097150	10.0.1.254	10.0.1.1	TCP	60	39863 → 1455 SYN] Seq=0 Win=1024 Len=0 MSS=1460
5500	231.097194	10.0.1.254	10.0.1.1	TCP	60	39863 → 3390 SYN] Seq=0 Win=1024 Len=0 MSS=1460
5501	231.097194	10.0.1.254	10.0.1.1	TCP	60	39863 → 9968 SYN] Seq=0 Win=1024 Len=0 MSS=1460
5502	231.097221	10.0.1.254	10.0.1.1	TCP	60	39863 → 62078 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
5503	231.097237	10.0.1.254	10.0.1.1	TCP	60	39863 → 49 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
5504	231.097260	10.0.1.254	10.0.1.1	TCP	60	39863 → 2005 SYN] Seq=0 Win=1024 Len=0 MSS=1460
5505	231.097275	10.0.1.254	10.0.1.1	TCP	60	39863 → 1247 SYN] Seq=0 Win=1024 Len=0 MSS=1460

Abbildung 18. TCP-Scan in Wireshark

Somit ergibt sich, dass folgende auftretende Merkmale einen Portscan in dieser Diplomarbeit definieren:

> Eine **Source-IP-Adresse** fragt innerhalb von **einer Minute** mindestens **500 verschiedene Ports** auf einem Ziel-Host mit der **gleichen Destination-IP-Adresse** ab. Hierbei wird lediglich auf **UDP- und TCP-Pakete** geachtet.

5.2 Umsetzung der Portscan-Erkennung mittels Skript

Zur Erkennung von Portscans wird ein selbstgeschriebenes Python-Skript verwendet[3]. Dieses besteht aus den drei Methoden: `on_start()`, `write_to_file()` und `find_portscan()`. Des Weiteren wird eine eigene **Portscan-Klasse** erstellt und verwendet. Dadurch können Portscans leichter identifiziert werden.

Class Portscan

Im Prozess des Durchsuchens werden alle potenziellen beziehungsweise auch alle tatsächlichen Portscans als Portscan-Objekte angelegt und mitgespeichert. Zu jedem einzelnen Objekt werden die folgenden Werte abgespeichert:

- Destination-IP-Adresse

[3] Das vollständige Skript ist als externe Datei „Portscan_Sniffer.py“ auf der CD des Bibliotheksexemplars hinterlegt.

- Liste aller Destination-Ports
- Source-IP-Adresse
- Protokoll (UDP/TCP)
- Start-Timestamp
- is_Portscan (default=False)

Damit die Objekte auch im Programm verwendet und verändert werden können, gibt es folgende verschiedene Methoden:

- **add_port(self, port)**: Dient dem Hinzufügen eines Ports zur Liste mit den Destination-Ports.
- **set_scan(self)**: Setzt den Wert, ob es sich um einen Portscan handelt, auf **`True`**.
- **__cmp__(self, other)**: Durch diese Methode können zwei Portscan-Objekte miteinander verglichen werden. Sie gelten als gleich, sobald ihre Source- & Destination-IP ident sind.
- **__str__(self)**: Ermöglicht das Ausgeben eines Portscan-Objekts als String. Dies wird für das Mitspeichern der Objekte benötigt.

on_start()

Diese Methode wird einmal zu Beginn der Skript-Ausführung aufgerufen. Durch sie wird die Datei **`scans.txt`** eingelesen, in der alle möglichen sowie tatsächlichen Portscans hineingeschrieben werden. Hierbei gilt, dass jeweils eine Zeile einem Objekt entspricht. Somit werden die einzelnen Zeilen eingelesen und die daraus erstellten Portscan-Objekte der Liste **`scan_list`** hinzugefügt.

write_to_file()

Mit Hilfe dieser Methode werden alle aktuellen Portscan-Kandidaten am Ende jedes durchsuchten PCAP-Files in die Datei **`scans.txt`** geschrieben. Hierzu wird die **`__str__()`**-Methode benötigt, um jedes einzelne Objekt als String in die Datei zu schreiben. Es wird jeweils ein Objekt pro Zeile gespeichert und die einzelnen Attribute wie in der **`__str__()`**-Methode definiert durch ein Semikolon getrennt.

Eine Zeile in der **`scans.txt`**-Datei kann somit wie folgt aussehen:

```
10.7.7.10;5000,5247,5246;10.7.7.37;17;2021-02-25 12:43:14.093761;False
```

Listing 12. Beispielzeile der Datei "scans.txt"

find_portscan()

In dieser Methode findet die eigentliche Portscan-Erkennung statt. Es werden alle PCAP-Dateien, die im definierten Ordner **`E:/PCAPs`** gespeichert sind, nacheinander durchgegangen. Hierbei spielt besonders die Liste **`scan_list`**, in der alle Portscan-Objekte gespeichert werden, eine große Rolle.

Im ersten Schritt werden nur jene Pakete beachtet, die den benötigten IP-Layer besitzen. Anschließend wird die Destination-IP-Adresse mit jenen der Portscan-Objekte, die in der zuvor erwähnten Liste gespeichert sind, verglichen. Ist die IP-Adresse noch nicht vorhanden, wird ein neues Portscan-Objekt mit den entsprechenden Werten des aktuellen Pakets der Liste hinzugefügt. Sollte die IP-Adresse bereits in einem Objekt vorhanden sein, so werden nun auch die Source-IP-Adresse sowie das Protokoll verglichen. Stimmen alle drei Werte mit einem vorhandenen Objekt überein, wird die vergangene Zeit zwischen dem Start-Timestamp und dem Timestamp des aktuellen Pakets herangezogen. Sofern weniger als 60 Sekunden vergangen sind und die Grenze von 500 Ports überschritten ist, wird das Objekt als gültiger Portscan deklariert und im definierten Format über die **`write_to_file()`**-Methode in eine Ergebnisdatei gespeichert. Sollte der Grenzwert unterschritten sein und der Port auch nicht in einer Destination-Portliste eines bestehenden Objekts enthalten sein, so wird der Port des aktuellen Pakets dem entsprechenden Objekt hinzugefügt.

Zuletzt wird in einem Durchlauf jedes Portscan-Objekt der zuvor erwähnten **`scan_list`** auf die Anzahl der Ports und die Dauer, die zwischen dem Start-Timestamp und dem Timestamp des aktuellen Pakets vergangen ist, überprüft. Sind bereits mehr als 60 Sekunden vergangen und handelt es sich um keinen Portscan, so wird das Objekt gelöscht. Weiters werden auch bereits als gültig deklarierte Portscans gelöscht, sofern sie schon länger als fünf Minuten vorhanden sind. Nachdem die PCAP-Files vom Programm analysiert worden sind, werden sie verschoben, damit sie nicht bei einem neuerlichen Skript-Aufruf ein zweites Mal eingelesen werden können.

Die Ausgabe in der Ergebnis-Datei kann folgendermaßen aussehen:

```
10.0.1.254;10.0.1.10;2020-12-29 20:11:28.449130
10.0.1.254;10.0.1.1;2020-12-29 20:11:28.448990
```

Listing 13. beispielhafte Portscan-Ergebnisse

5.3 Automatisierung

Das Portscan-Skript wird auf dem Diplomarbeits-Cluster auf einer eigenen virtuellen Windows-Maschine ausgeführt. Damit das Skript automatisch ausgeführt wird, ist ein entsprechender *ScheduledJob* zu konfigurieren, der das Skript, also folgende Befehlszeile, aufruft:

```
python E:\Scripts\Portscan\Portscan_Sniffer.py
```

Listing 14. PowerShell: Skript-Aufruf

Das Skript arbeitet anschließend alle Dateien im Ordner **`E:\PCAPs`** durch und sucht nach den definierten Portscan-Merkmalen. In den soeben erwähnten Ordner werden automatisch die PCAP-Dateien von der *Share-VM* kopiert. Dies erfolgt auch über einen *ScheduledJob*, wie im Kapitel „Client-VMs“ beschrieben. Für die Automatisierung des Skript-Aufrufs wird nun ein *ScheduledJob* benötigt. Dieser versucht alle fünf Minuten, das Skript

auszuführen. Sollte das Skript noch aktiv sein, verzögert sich durch den *ScheduledJob* automatisch der nächste Aufruf und es kommt zu keinem Fehler.

Für das Erstellen des *ScheduledJobs* werden im ersten Schritt die folgenden benötigten Variablen definiert:

- Der **Trigger** versucht, das Skript einmal alle fünf Minuten zu starten.

```
$trigger = New-JobTrigger -Once -At (Get-Date) -RepetitionDuration
([System.TimeSpan]::MaxValue) -RepetitionInterval (New-TimeSpan -Minutes
5)
```

Listing 15. PowerShell: Trigger-Variable Definition

- Ein **Skriptblock**, in dem der eigentliche Aufruf des Portscan-Skripts angegeben ist.

```
$scriptblock = [scriptblock]::Create(“python
E:\Scripts\Portscan\Portscan_Sniffer.py“)
```

Listing 16. PowerShell: Skriptblock-Variable Definition

- Weitere **Optionen**, die das Verhalten des *ScheduledJob* bestimmen.

```
$options = New-ScheduledJobOption -RunElevated -ContinueIfGoingOnBattery
– StartIfOnBattery
```

Listing 17. PowerShell: Optionen-Variable Definition

Mit den erstellten Variablen wird nun der *ScheduledJob* eingerichtet. Hierzu muss ein Name angegeben werden, in diesem Fall **`Start Portscan-Skript`**. Für die Befehlsoptionen werden die zuvor definierten Variablen angegeben. Zuletzt wird für den Parameter *Credential* der Benutzername „Specto“ übergeben. Beim Erstellen wird man daraufhin aufgefordert, das entsprechende Passwort einzugeben.

```
Register-ScheduledJob -Name “Start Portscan-Skript“ -ScriptBlock
$scriptblock -Trigger $trigger -ScheduledJobOption $options -Credential
(Get-Credential Specto)
```

Listing 18. PowerShell: Erstellung des ScheduledJobs

Mit dem *PowerShell-Cmdlet* **`Get-ScheduledJob`** kann der erstellte *ScheduledJob* überprüft werden. Neben dem *ScheduledJob* für das Verschieben der PCAP-Dateien ist nun auch der *ScheduledJob* für das Starten des Portscan Skripts vorzufinden.

```
PS C:\Users\Specto> Get-ScheduledJob

Id         Name            JobTriggers     Command                                 Enabled
--         ----            -----------     -------                                 -------
1          Collect PCAPs   1               Copy-Item -Path //10.70.8.1/share/PCA... True
2          Start Portsc... 1               python E:\Scripts\Portscan\Portscan_S... True
```

Abbildung 19. Überprüfen der konfigurierten ScheduledJobs in der PowerShell

Nach der Konfiguration wird das Skript alle fünf Minuten ausgeführt. Befinden sich im PCAP-Ordner **`E:\PCAPs`** PCAP-Dateien, so werden alle nacheinander analysiert und entsprechend der weiter oben angeführten Beschreibung nach Portscans durchsucht. Sollte

das Skript abstürzen oder für längere Zeit nicht aufgerufen worden sein, so stellt dies kein Problem dar, da das Skript nebenbei alle möglichen Portscan-Objekte in der **`scans.txt`**-Datei mitspeichert.

6 Erfassung der Netzwerkverbindungen

Damit Daten im Rahmen der Diplomarbeit „Specto“ verarbeitet werden können, müssen diese zunächst erfasst werden. Für die Kategorisierung und Erstellung von Statistiken sind hierbei Informationen zu Netzwerkverbindungen wichtig. Genauer werden folgende Informationen benötigt:

- IP-Adresse des Quellgeräts
- IP-Adresse des Zielgeräts
- Port des Quellgeräts
- Port des Zielgeräts
- Transportprotokoll der Netzverbindung
- Zeitstempel

6.1 NetFlow

Um die oben genannten Informationen aus einem Netzwerk zu erfassen, kann *NetFlow* verwendet werden.

NetFlow ist eine von Cisco entwickelte Technik, die es einem Router oder Layer-3-Switch ermöglicht Netzwerkverbindungen lokal zwischenzuspeichern. Die zwischengespeicherten Datenströme, auch *Flow Records* genannt, können anschließend als Paket an eine Sammelmaschine gesendet werden. Diese Sammelmaschine liest die Datenströme mit einem Programm ein und wertet sie, meist graphisch, aus. Durch die Analyse kann sich ein Administrator anschließend ein Bild von der Auslastung des Netzes machen. (vgl. SolarWinds o. J.)

NetFlow besitzt außerdem verschiedene Versionen, wobei Version 9 als offener Standard in *RFC 3945* beschrieben wird.

6.1.1 Aufbau

Damit NetFlow in einem Netz implementiert werden kann, werden zumindest folgende Komponenten im Zielnetz benötigt:

- Ein Gerät, das Netzwerkverkehr erzeugt (Traffic-Generator)
- Ein Gerät, auf dem später die Datenströme gesammelt werden (Collector)
- Ein *NetFlow*-fähiger Router oder ein *NetFlow*-fähiger Layer-3-Switch, der Flow Records erstellt

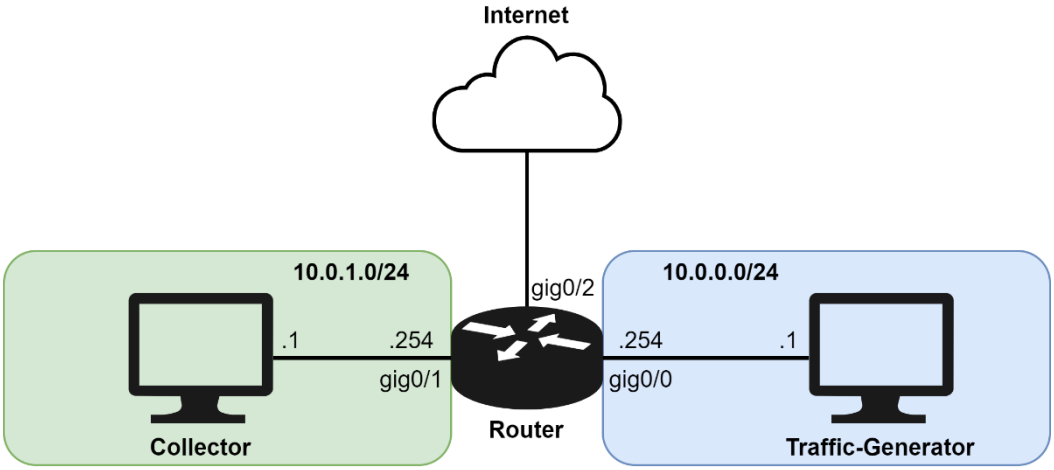


Abbildung 20. Minimale NetFlow-Umgebung

Somit könnte ein Netzwerk mit den Mindeststandards für die *NetFlow*-Implementation wie in Abbildung 20 aussehen.

Vor der Konfiguration muss außerdem eine NetFlow-Version ausgewählt werden. Im Rahmen der Diplomarbeit „Specto“ wurde ausschließlich Version 9 verwendet. Diese Version ist *template-based*. Das heißt, es wird zunächst eine Struktur aus Datenfeldern erstellt, nach der die Datenströme aufgebaut werden sollen. Diese Struktur wird anfangs dem *Collector* mitgeteilt und macht es so einfacher neue Features hinzuzufügen, ohne die aktuelle Implementation zu zerstören. (vgl. Cisco 2011)

6.1.2 Konfiguration

Damit nun Daten im Netz erfasst werden können, muss *NetFlow* auf einem Router bzw. einem Layer-3-Switch konfiguriert werden. Dieser Router sollte jedoch kein x-beliebiger Router sein, sondern sollte dem Anwendungsfall entsprechend eine bestimmte Position im Netz besitzen.

Im Falle der Diplomarbeit „Specto“ soll Netzwerkverkehr in das Schulnetzwerk gehend und aus dem Schulnetzwerk hinausgehend ermittelt werden. Hierzu bietet sich der Router an, der die Außengrenze des Schulnetzwerkes bildet bzw. direkt mit dem *ISP* verbunden ist. Genauer gesagt wird *NetFlow* in dieser Diplomarbeit auf der Firewall konfiguriert.

Auf dieser wird zu Beginn ein *Flow Record* wie folgt in der CLI konfiguriert:

```
conf t
flow record NetFlow
match ipv4 source address
match ipv4 destination address
match ipv4 protocol
match transport source-port
match transport destination-port
match ipv4 tos
match interface input
```

```
collect interface output
collect counter bytes
collect counter packets
collect timestamp sys-uptime first
collect timestamp sys-uptime last
```

Listing 19. Cisco CLI Flow Record

Später erstellte *Flow Records* enthalten somit die Felder, die durch den Flow Record *NetFlow* konfiguriert wurden. Das sind, wie man am obigen Code sehen kann, alle erforderlichen Informationen, die in *6 Erfassung der Netzwerkverbindungen* aufgezählt wurden. Zusätzlich könnten auch noch die Nummern der *Autonomen Systeme* erfasst werden, dies ist jedoch für diesen Teilbereich von „Specto“ nicht erforderlich.

Im nächsten Schritt wird ein *Exporter NetFlowExporter* konfiguriert. Bei der Konfiguration des Exporters ist zum ersten Mal die Angabe einer Version erforderlich. Wie zuvor erwähnt, wird Version 9 implementiert.

Die Konfiguration auf der Firewall würde im Beispiel aus Abbildung 20 wie folgt aussehen:

```
flow exporter NetFlowExporter
destination 10.0.1.1
source gig0/1
transport UDP 2055
export-protocol netflow-v9
template data timeout 30
```

Listing 20. Cisco CLI NetFlow-Exporter

In der obigen Konfiguration wurden eingestellt, welche IP-Adresse der Kollektor im Netzwerk besitzt, aus welchem Interface der Firewall die gesammelten Datenströme gesendet werden sollen und welcher UDP-Port zur Übertragung verwendet werden soll.

Zusätzlich wurde konfiguriert, dass *NetFlow*-Version 9 verwendet wird und die definierte Struktur alle 30 Sekunden ausgesendet werden soll.

Anschließend wird ein Monitor NetFlowMonitor erstellt. Hier ist zu beachten, dass für **`record`** und **`exporter`** die gleichen Namen wie von den zuvor erstellten Elementen benutzt werden.

```
flow monitor NetFlowMonitor
record NetFlow
exporter NetFlowExporter
cache timeout active 60
cache timeout inactive 15
exit
int gig0/0
ip flow monitor NetFlowMonitor input
```

```
ip flow monitor NetFlowMonitor output
```

Listing 21. Cisco CLI NetFlow-Monitor

Durch die Konfiguration des Monitors wird das Zwischenspeichern der Datenströme eingestellt. Außerdem wird mit Angabe des Monitors im Interface eingestellt, dass nur Daten von und zu diesem Interface erfasst werden.

Zu guter Letzt werden nur noch die Interfaces des Routers als ingress/egress deklariert.

```
ip flow-export version 9
int gig0/0
ip flow ingress
int gig0/1
ip flow egress
int gig0/2
ip flow egress
exit
ip flow-export destination 10.0.1.1 2055
```

Listing 22. Cisco CLI NetFlow-Monitor

Nach diesen Schritten ist die Konfiguration von NetFlow auf der Firewall abgeschlossen. Nachdem Datenverkehr von dem Traffic-Generator erzeugt wurde, sollte die Firewall beginnen diesen zwischenzuspeichern. (vgl. Cisco o. J.; SolarWinds 2018)

Diese Funktion kann mit folgendem Befehl überprüft werden:

```
show ip flow export
```

Listing 23. Cisco CLI Anzeigen der exportierten Flows

Das Ergebnis des Befehls kann in Abbildung 21 betrachtet werden.

```
Router#sh ip flow export
Flow export v9 is enabled for main cache
  Export source and destination details :
  VRF ID : Default
    Destination(1)  10.0.1.1 (2055)
  Version 9 flow records
  387 flows exported in 17 udp datagrams
  0 flows failed due to lack of export packet
  0 export packets were sent up to process level
  0 export packets were dropped due to no fib
  0 export packets were dropped due to adjacency issues
  0 export packets were dropped due to fragmentation failures
  0 export packets were dropped due to encapsulation fixup failures
```

Abbildung 21. Ausgabe der NetFlow-Konfiguration

6.1.3 Collector

Zum Sammeln und Auswerten der gespeicherten Flow Records wird ein Collector benötigt. Dieser Collector besteht aus einem Gerät, auf dem ein, meist kostenpflichtiges, Programm installiert wird. Dieses Programm bietet, je nach Preisklasse, unterschiedliche Features, um die empfangenen Datenströme auszuwerten.

6.1.3.1 Plixer Scrutinizer

Der erste Collector, der für die Diplomarbeit „Specto“ in Frage kam, war *Scrutinizer* von *Plixer*. Um diesen Collector zu implementieren, wird die Software *Scrutinzer* von der Plixer-Website erworben. Hierbei ist zu beachten, dass Scrutinizer ein kostenpflichtiges Produkt ist. Im Rahmen dieser Diplomarbeit wurde die kostenlose 30-Tage-Version mit allen notwendigen Funktionen getestet.

Nach der Installation wird lediglich eingestellt an welchen Ports NetFlow-Pakte zu erwarten sind, falls diese nicht unter den standardmäßig benutzten (2055, 2056, 4432, 4739, 9995, 9996 und 6343) sind.

Anschließend sind die empfangenen und analysierten Daten über eine webbasierte Benutzeroberfläche zu begutachten.

6.1.3.2 SolarWinds Real-time NetFlow Analyzer

Der *Real-time NetFlow Analyzer* von *SolarWinds* ist ähnlich zu Scrutinizer. Die einzigen Unterschiede sind, dass dieses Produkt völlig kostenlos und nicht webbasiert ist.

Nach der Installation wird bei dem *Real-time NetFlow Analyzer* das Netzwerk-Interface angegeben, an welchem die NetFlow-Pakete erwartet werden. Danach wird der Flow Capture gestartet und die empfangenen Datenströme werden sofort analysiert und in der Benutzeroberfläche durch Diagramme dargestellt. Die graphische Analyse von aktuellen Netzwerkverbindungen und deren Anteil im Netz sieht beispielsweise wie in Abbildung 22 aus.

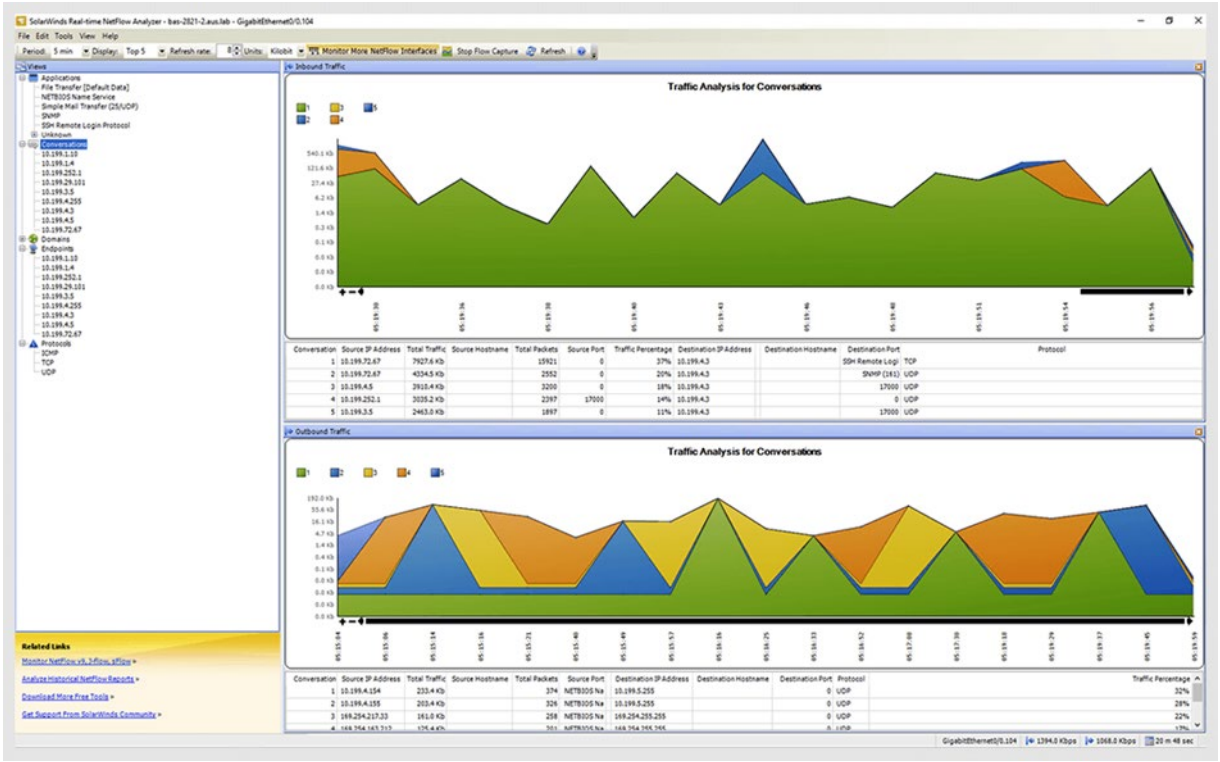


Abbildung 22. Oberfläche das SolarWinds Collectors
Quelle: https://www.solarwinds.com/-/media/solarwinds/swdcv2/free-tools/real-time-netflow-analyzer/images/rna-traffic-analysis.ashx?rev=641498081ccc47219d8e505af0455b16

6.1.3.3 Eigener Collector (Wireshark & Scapy)

Auch wenn kommerzielle NetFlow-Collectoren eine breite Auswahl an Möglichkeiten bieten sind sie aus zwei Gründen für diese Diplomarbeit nicht zu gebrauchen.

- Erstens sind der Großteil der NetFlow-Collectoren kostenpflichtig. Der käufliche Erwerb dieser Collectoren würde also den Rahmen des Diplomarbeitsbudgets sprengen und das Implementieren der kostenlose 30-Tage-Variante ist langfristig gesehen nicht von Vorteil.
- Zweitens, und viel wichtiger, erfüllen alle kommerziellen NetFlow-Collectoren nicht den Zweck, der von „Specto“ benötigt wird. Alle Collectoren analysieren sofort die empfangenen Daten und stellen diese graphisch dar. Hier gibt es wenig Möglichkeiten auf die Datenbank der Produkte, und somit auf die Rohdaten, zuzugreifen. Genau diese Rohdaten werden aber von der Diplomarbeit „Specto“ benötigt.

Daher bietet es sich an einen eigenen NetFlow-Collector zu erstellen. Dadurch kann entschieden werden, was mit den Daten geschehen soll, bevor diese analysiert werden. Außerdem bleiben die Daten in vertraulichen Händen und werden an keine Dritten weitergegeben.

Die folgenden Schritte wurden im Rahmen von „Specto“ auf einer *Linux Mint*-Maschine durchgeführt.

Um einen eigenen *NetFlow-Collector* zu realisieren, wird in diesem Teilbereich zunächst von dem kostenlosen Paketinspektions-Tool *Wireshark* Gebrauch gemacht. Wireshark kann Netzwerkverkehr eines Interfaces aufzeichnen, graphisch darstellen und in Dateien abspeichern.

Eines der häufig verwendeten Dateiformate zum Speichern ist *.pcap*. PCAP ist eine *API*, die es erlaubt Netzwerkverkehr mitzuschneiden oder auch gespeicherte Netzwerkdaten mit dieser zu interpretieren. (vgl. ReviverSoft o. J.)

Nun wird zunächst Wireshark auf der Linux-Maschine installiert. Während der Installation wird die Frage gestellt, ob nur der Root-User Wireshark verwenden darf. Dies muss verneint werden. Zusätzlich muss der aktuelle User zur Wireshark-Gruppe hinzugefügt werden. Dies wird mit folgendem Befehl realisiert:

```
sudo usermod -a -G wireshark "$USER"
```

Listing 24. Linux Shell User zu Wireshark-Gruppe hinzufügen

Um nun Wireshark in Verbindung mit NetFlow benutzen zu können, muss auch die Firewall dementsprechend konfiguriert werden. Das heißt, die Firewall muss die CFLOW-Pakete in Richtung Linux-Maschine erlauben. Im Falle dieser Diplomarbeit wurde der UDP-Port 2055 benutzt.

```
sudo iptables -A INPUT -p udp --dport 2055 -j ACCEPT
systemctl start ufw
```

Listing 25. Linux Shell Firewall konfigurieren und starten

Damit ist die Vorbereitung für Wireshark abgeschlossen. Um PCAP-Dateien zu erstellen, kann Wireshark mit der Angabe einiger Attribute in der Shell angegeben werden. Zum Beispiel kann Wireshark

- für das Interface ens33 (-i)
- mit einem Filter (-f),
- einem Ringpuffer (-b) und
- einer automatisch erstellten Zieldatei (-w)
- sofort (-k)

gestartet werden.

```
wireshark -i ens33 -k -f 'udp port 2055' -b duration:60 -w
```

Listing 26. Linux Shell Wireshark starten

Nachdem erste PCAP-Dateien erstellt wurden, wird für diese Diplomarbeit das Python-Package *Scapy* verwendet, um diese auszuwerten.

Scapy ist sowohl ein Paketinspektions-, als auch ein Paketmanipulationstool. Es bietet neben dem Erstellen und Dekodieren großer Mengen an Paketen, dem Erfassen und dem Abgleichen von Daten viele weitere Funktionen. (vgl. Scapy o. J. a)

Damit *Scapy* im Rahmen von „Specto“ als *Collector* verwendet werden kann, werden nur folgende zwei Funktionen von *Scapy* benötigt:

- Auslesen von *PCAP*-Dateien
- Defragmentieren in NetFlow-Datenfelder

Damit PCAP-Dateien mittels *Scapy* ausgelesen werden können, werden einfachheitshalber alle Funktionen von *Scapy* importiert. Dies sieht im Programm-Code wie folgt aus:

```
from scapy.all import *
```

Listing 27. Python Scapy importieren

Anschließend muss für das Defragmentieren der Daten in *NetFlow*-Datenfelder zusätzlich das untergeordnete Layer-Package für *NetFlow* von *Scapy* importiert werden:

```
from scapy.layers.netflow import *
```

Listing 28. Python Scapy Layer-Package importieren

Im Programmcode wird die Funktion **`writeFlows()`** definiert. Dieser Funktion werden zwei Attribute mitgegeben.

1. die Quell-PCAP-Datei, aus der die Flow Records entnommen werden sollen.
2. die Ziel-Datei, in die die ausgewerteten Daten im intern festgelegten CSV-Format gespeichert werden sollen.

Wird nun eine gültige PCAP-Datei angegeben, so wird im Programmcode diese geöffnet und in eine Variable gespeichert. Durch diese Variable wird anschließend so lange durchiteriert, bis das Ende des Pakets erreicht ist. Dies sieht verkürzt so aus:

```
packets = sniff(offline=open(pcap, "rb"), session=NetflowSession)
for packet in packets:
    if packet.haslayer(NetflowDataflowsetV9):
        for flowRecord in packet[NetflowDataflowsetV9].records:
```

Listing 29. Python Iteration durch jeden Flow

Wie dem Code entnommen werden kann, wird bei jedem Paket überprüft, ob ein Flow-Set der Version 9 vorhanden ist.

In dieser Schleife werden anschließend nach und nach alle Datensätze erfasst und in eine Datei geschrieben. Um auf die Datenfelder von NetFlow zugreifen zu können, muss lediglich auf vordefinierte Variablen des NetFlow-Flow-Sets zugegriffen werden. (vgl. Scapy o. J. b)

Beispielsweise sieht der Zugriff auf Quell- und Ziel-IP-Adresse in der Schleife wie folgt aus:

```
srcip = flowRecord.IPV4_SRC_ADDR
dstip = flowRecord.IPV4_DST_ADDR
```

Listing 30. Python Zugriff auf IP-Adressen von Flows

Hiermit ist die Quell-IP-Adresse in der Variable **`srcip`** und die Ziel-IP-Adresse in der Variable **`dstip`** gespeichert. Dieser Vorgang wird für alle in *Erfassung der Netzwerkverbindungen* deklarierten notwendigen Informationen getätigt.

Damit werden diese Datensätze, wie zuvor erwähnt, im CSV-Format in die angegebene Zieldatei gespeichert. Die Reihenfolge ist dabei intern festgelegt:

```
sourceip;destinationip;sourceport;destinationport;protocol;timestamp
```

Listing 31. Reihenfolge der Ausgabe

Die fertigen Datensätze sehen in der Zieldatei wie folgt aus:

```
10.0.0.7;172.217.20.14;65439;443;UDP;19.01.2021 19:28:18
10.0.0.7;10.0.0.138;64777;53;UDP;19.01.2021 19:28:18
```

Listing 32. Beispiel für Ausgabe

Um das Benutzen des Python-Skripts in einer Shell zu vereinfachen, wurde zusätzlich *argparse* implementiert. Als Variablen von *argparse* wurden nur die erforderliche Quell- und Zieldatei und zusätzlich die Funktion *verbose* für eine hilfreiche Ausgabe hinzugefügt.

Somit kann das fertige Collector-Skript[4] beispielsweise in einer Shell wie folgt aufgerufen werden:

[4] Das vollständige Skript ist als externe Datei „NetFlow_Collector.py“ auf der CD des Bibliotheksexemplars hinterlegt.

```
python3 Desktop/NetFlow-Collector.py -s netflow_00001_20201217192726 -d
netflow01.txt
```

Listing 33. Linux Shell Aufruf des Skripts mit argparse-Variablen

6.2 Ohne NetFlow

Wie zuvor erwähnt wird NetFlow für den Gebrauch der Diplomarbeit „Specto“ auf der zentralen Firewall implementiert. Bei mehr als 1000 SchülerInnen und vielen Lehrkräften mit mehr als einem Elektrogerät befinden sich sehr viele Geräte im Schulnetzwerk. Das heißt, für den normalen Gebrauch benötigt diese Firewall sehr viel Leistung.

Wird nun zusätzlich NetFlow auf dieser konfiguriert, wird durch das Zwischenspeichern der Datenströme eine höhere Leistung benötigt. Daher wäre NetFlow langfristig gesehen nicht rentabel für das gesamte Schulnetzwerk.

6.2.1 Aufbau

Das zuvor genannte Python-Package *Scapy* ist wie erwähnt sehr umfangreich. Mit der Hilfe von *Scapy* können neben der Auswertung von *NetFlow*-Ebenen auch die normalerweise vorhandenen IP-Ebenen der Pakete ausgewertet werden. Somit können jegliche Datenströme auch ohne NetFlow erfasst werden.

Die Mindeststandards für das Netzwerk bleiben jedoch wie in Abbildung 20. Es werden zumindest folgende Geräte benötigt.

- Ein Gerät, das Netzwerkverkehr erzeugt (Traffic-Generator)
- Ein Gerät, auf dem später die Datenströme gesammelt werden (Collector)
- Ein Router oder ein Layer-3-Switch

In dieser Variante der Implementation kann auf die Konfiguration von NetFlow am Router verzichtet werden. Jedoch muss stattdessen ein *Mirror-Port* auf dem Router konfiguriert werden, der alle empfangenen Pakete an das Interface in Richtung Collector weiterleitet. Im Beispiel von Abbildung 20 wäre dieses Interface Gigabit Ethernet 0/1.

6.2.2 Collector

Um nun die Datenströme des gesamten Schulnetzwerkes zu erfassen, wird ein Python Skript mit Scapy benutzt, ohne NetFlow zu konfigurieren.

Damit die vom Mirror-Port kommenden Datenpakete erfasst werden, muss wie in *6.1.3.3 Eigener Collector (Wireshark & Scapy)* Wireshark auf einer Linux-Maschine installiert und eingerichtet werden. Anschließend kann der gleiche Befehl benutzt werden, um empfangene Pakete in PCAP-Dateien abzuspeichern.

```
wireshark -i ens33 -k -f 'udp port 2055' -b duration:60 -w
```

Listing 34. Linux Shell Wireshark starten

Danach wird wie in *6.1.3.3 Eigener Collector (Wireshark & Scapy)* das Modul Scapy und zusätzlich das Layer-Package *inet* importiert.

```
from scapy.all import *
from scapy.layers.inet import *
```

Listing 35. Python Scapy und Layer-Package importieren

Anstatt nun *Flow Records* von *NetFlow* zu betrachten, werden einfach die notwendigen Daten, aus den Paketen selbst ausgewertet. Hierzu werden Werte aus den Scapy-Ebenen IP und die des verwendeten Protokolls in Variablen gespeichert. (vgl. ProgramCreek o. J.)

Das Speichern der IP-basierten Daten sieht im Code beispielsweise wie folgt aus:

```
if packet.haslayer(IP):
    srcip = packet[IP].src
    dstip = packet[IP].dst
    proto = PROTOCOLS[packet[IP].proto]
else:
    srcip = None
    dstip = None
    proto = None
```

Listing 36. Python Zugriff auf IP-Adressen ohne NetFlow

In diesem Teil des Codes kann gesehen werden, dass die erforderlichen Variablen nicht gesetzt werden, wenn das Paket keine IP-Ebene enthält. Dies kommt im Laufe des Programm-Codes zu nutzen.

Vor dem Beschreiben der Zieldatei wird überprüft, ob keine der relevanten Variablen ungültig ist. Ist eine der Variablen ungültig, so ist dieser Datenstrom nicht relevant und wird nicht in die Zieldatei geschrieben. Schlussendlich wird eine Datei mit Datenströmen wie in Listing 31 vom Python-Skript[5] erstellt.

6.3 Implementation im Schulnetzwerk

Da die *Packet Inspection* im Rahmen dieser Diplomarbeit mehrmals benötigt wird, wird diese ausgelagert. Die *Packet Inspection* findet daher im Schulnetzwerk nicht auf dem Collector, sondern auf einer von Cerebrum bereitgestellten virtuellen Maschine, statt. Daher muss sich der Collector im Schulnetz nur um die Verarbeitung der PCAP-Dateien kümmern. Dies kann auch in Abbildung 1 bzw. Abbildung 2 betrachtet werden.

6.3.1 Automatisierung

Um die Aufgaben des Collectors zu bewerkstelligen, muss dieser automatisiert werden. Hierzu muss der Collector zunächst PCAP-Dateien aus einem lokalen Ordner mithilfe des in

[5] Das vollständige Skript ist als externe Datei „Collector.py“ auf der CD des Bibliotheksexemplars hinterlegt.

6.2.2 Collector beschriebenen Python-Skripts verarbeitet und in Dateien gespeichert werden. Diese Dateien werden danach auf eine externe Freigabe kopiert und auf dem Collector lokal gelöscht.

Um diese Abläufe nun in einer Linux-Umgebung zu automatisieren, wird ein Shell-Skript namens **`.processData.sh`** wie folgt erstellt:

```
nano Schreibtisch/.processData.sh
```

Listing 37. Linux Shell-Skript erstellen

Dieses Skript läuft in einer unendlichen Schleife und „schläft" nach einem Durchlauf 60 Sekunden. In einem Durchlauf wird jede PCAP-Datei, die sich im Ordner PCAPs befindet, mithilfe des Python-Skripts verarbeitet und in eine eigene Datei geschrieben. Der Name dieser Datei enthält das Datum, damit die Datei einzigartig ist. Dieser Ablauf sieht in der Schleife wie folgt aus:

```
for file in $(ls /home/specto/Schreibtisch/PCAPs)
    do
        date=$(date '+%Y-%m-%d_%H-%M-%S')
        python3 /home/specto/Schreibtisch/Collector.py -s
/home/specto/Schreibtisch/PCAPs/$file -d
/home/specto/Schreibtisch/netflow_$date.txt
```

Listing 38. Linux PCAP-Dateien in Schleife auswerten

Die erstellte Datei wird anschließend auf den externen Share der *Share-VM* (10.70.8.1) in den Ordner *NetFlow* kopiert. Anschließend wird die erstellte Datei genau wie die PCAP-Dateien lokal gelöscht. Dieser Vorgang sieht folgendermaßen aus:

```
smbclient //10.70.8.1/share gansgeheim123! -c 'cd NetFlow;put
/home/specto/Schreibtisch/netflow_'$date'.txt netflow_'$date'.txt'
        rm /home/specto/Schreibtisch/netflow_$date.txt
        rm /home/specto/Schreibtisch/PCAPs/$file
```

Listing 39. Linux Erstellte Textdateien auf Share kopieren und lokal löschen

Wie zuvor erwähnt wird nach diesem Zyklus 60 Sekunden gewartet und der Ablauf beginnt von neuem. Damit muss dieses Skript nur einmal aufgerufen werden. Hierzu bietet sich ein *Cronjob* an. Um einen Cronjob zu erstellen öffnet man zunächst die Crontabelle, kurz Crontab. (vgl. Stetic o. J.)

Das Öffnen der Crontab sieht wie folgt aus:

```
crontab -e
```

Listing 40. Linux Crontab öffnen

Nun wird ein Eintrag in der Crontab erstellt. Dieser Eintrag startet das zuvor erstellte Shell-Skript bei jedem Reboot des Collectors. Dazu wird eine Zeile mit **`@reboot`** unten in crontab angefügt:

```
@reboot bash /home/specto/Schreibtisch/.processData.sh
```

Listing 41. Linux Shell-Skript bei Reboot automatisch starten

Nun wird bei jedem Start der Maschine das Shell-Skript automatisch ausgeführt und PCAP-Dateien werden automatisch verarbeitet und verschoben.

7 Kategorisierung

Damit erfasste Daten aus *6 Erfassung der Netzwerkverbindungen* in grobe Bereiche eingeteilt werden können, arbeitet die Diplomarbeit „Specto“ mit einer eigenen Kategorisierung. Die theoretische Einteilung in Kategorien wird von Cerebrum verfasst und basiert auf dem Transportprotokoll und dem dazugehörigen Port.

Da es eine standardisierte Liste der IANA an UDP/TCP-Ports zu Protokollen von Diensten gibt, kann Netzwerkverkehr anhand dieser Merkmale grob eingeteilt werden. (vgl. IANA 2021)

Ein Auszug aus der Kategorisierung sieht wie folgt aus:

Tabelle 1. Einteilung in Kategorien

Kategorie	Merkmale (Vertreter der Kategorie, wenn…)
Webtraffic	TCP Port 80 (HTTP) TCP Port 443 (HTTPS) UDP Port 443 (Chrome verwendet auch UDP) TCP Port 3128 (HTTP)
MusikStream	TCP Port 80001 (SHOUTcast) UDP Port 57621 (Spotify)

7.1 Praktische Umsetzung

Damit im Rahmen dieser Diplomarbeit Netzwerkverkehr tatsächlich in Kategorien eingeteilt werden, muss das Python-Skript aus *6.2 Ohne NetFlow* erweitert werden. Konkret wird die Methode **`getCategory()`** erstellt, die das Transportprotokoll und den Port als *Tupel* mitgeliefert bekommt.

Die Methode **`getCategory()`** wird nach dem Setzen der Quell- und Ziel-IP-Adresse aufgerufen. Dies sieht im Code wie folgt aus:

```
if srcip is not None and dstip is not None:
    [...]
    elif not ipaddress.ip_address(srcip).is_private:
        category = getCategory((proto, srcport))
    elif not ipaddress.ip_address(dstip).is_private:
        category = getCategory((proto, dstport))
    [...]
```

Listing 42. Python getCategory() aufrufen

In Listing 42 wird zuerst nachgesehen, ob die Quell- und Ziel-IP-Adresse gesetzt sind. Danach wird in einer weiteren Kondition überprüft, ob die Quell- bzw die Ziel-Adresse öffentlich ist. Nur, wenn die IP-Adresse öffentlich ist, ist der Port des Netzwerkverkehrs relevant, da hier die von der IANA festgelegten Ports verwendet werden.

Innerhalb der **`getCategory()`**-Methode werden die mitgelieferten Variablen benutzt, um als Schlüssel für ein *Dictionary* zu dienen. Sind die Variablen ein Schlüssel, der im *Dictionary* vorhanden ist, so liefert das *Dictionary* einen Wert zurück. Im Falle „Specto" entspricht dieser Wert einer konkreten Kategorie. Die Funktion des Dictionaries ersetzt das *Switch*-Statement aus Java. (vgl. StackOverflow o. J.)

Dieser Vorgang sieht für das Beispiel aus Tabelle 1 im Code gekürzt wie folgt aus:

```
CATEGORY = {
    ("TCP", 80): "Webtraffic",
    ("TCP", 443): "Webtraffic",
    ("UDP", 443): "Webtraffic",
    ("TCP", 3128): "Webtraffic",
    ("TCP", 80001): "Musik-Stream",
    ("UDP", 57621): "Musik-Stream"
}
[...]
def getCategory(characteristics):
    return CATEGORY.get(characteristics, None)
```

Listing 43. Python Kategorie aus Dictionary auslesen

Wird eine Netzwerkverbindung mit dem Transportprotokoll TCP und dem Port 443 erfasst, so liefert die Methode **`getCategory()`** den *String Webtraffic* zurück. Wird eine Netzwerkverbindung erkannt, zu denen kein Eintrag im *Dictionary* vorhanden ist, so liefert **`getCategory()`** *None* zurück.

Die ermittelte Kategorie wird anschließend im *Collector* an die Textzeile aus Listing 32 hinzugefügt und gemeinsam der Share-VM übermittelt.

8 Datenbanken

Das Speichern von Daten ist ein wesentlicher Bestandteil der Diplomarbeit „Specto“. Diese Daten unterteilen sich in Echtdaten sowie Statistiken und werden in Datenbanken zentralisiert gespeichert und verwaltet.

Die Echtdaten dienen der Medientechnik zur Visualisierung der Daten im Schulgebäude sowie auf der Weltkarte. Um das Visualisieren weiters bewerkstelligen zu können, werden Informationen zu den Klassen sowie Ländern gespeichert. Daten hierzu werden sowohl von „Nexus“ als auch von „Cerebrum“ bereitgestellt.

Die Statistiken ergeben sich aus den gesammelten Echtdaten. Diese werden, wenn möglich einer Klasse beziehungsweise den IP-Adressen zugeordnet. Statistiken ermöglichen es dem Netzwerkadministrator auf einem Blick zu erkennen, was in seinem Netzwerk an diesem Tag, in diesem Monat oder Jahr vorgegangen ist.

8.1 Echtdaten-Datenbank

In der Echtdaten-Datenbank sind alle noch nicht weiter verarbeiteten Daten gespeichert. Diese werden anschließend für die Erstellung der Statistiken sowie für die Bereitstellung an das Medientechnik-Partnerteam weiterverwendet.

Die Echtdaten werden in einer *NoSQL-Datenbank* gespeichert und werden laufend eingetragen. Es handelt sich dabei um MongoDB. Für diese sind entsprechende *JSON*[6]*-ähnliche* Dokument-Strukturen erstellt. Mittels eines Python-Skripts können die Daten aus CSV-Dateien eingelesen werden.

8.1.1 NoSQL und MongoDB

Im Gegensatz zu *relationalen Datenbanken* arbeiten *NoSQL-Datenbanken* nicht tabellenorientiert, sondern beispielsweise dokumentorientiert. Oft kann man zusammenhängende Daten einfacher speichern, da diese nicht zwischen Tabellen aufgeteilt werden müssen. Es können verwandte Daten in einem einzigen Dokument verschachtelt werden. Dadurch weisen diese Datenbanken ein einfacher zu verwaltendes Datenmodell als *relationale Datenbanken* auf. (vgl. MongoDB o. J. a)

Anders gesagt bedeutet das:

> NoSQL steht für „Not only SQL“ und bezeichnet Datenbanksysteme, die einen nicht-relationalen Ansatz verfolgen. Diese Datenbanken, denen verschiedene Datenbankmodelle zugrunde liegen können, sind horizontal skalierbar und lassen sich für Big-Data-Anwendungen einsetzen. (Luber 2017a)

[6] JavaScript Object Notation: einfach zu lesendes Datenformat

Bei MongoDB handelt es sich um die Datenbank, welche schlussendlich für das Speichern der Echtdaten ausgewählt wurde. Diese bietet die Möglichkeit, JSON-Dokumente möglichst effizient speichern zu können. Abfragen können mittels *JSON-Formats* getätigt werden und es werden eine Reihe von Aggregationen für die spätere Erstellung der Statistiken geboten. Weiters ist es wichtig, die Daten dem Zeitpunkt nach gefiltert abfragen zu können. Beispielsweise ist es von Bedeutung, sämtliche Verbindungen des heutigen Tages abfragen zu können, um diese weiterverarbeiten zu können.

8.1.2 MongoDB mit Python

Bei PyMongo handelt es sich um eine Library für Python. Diese kann für die verschiedenen Zugriffe auf MongoDB genutzt werden. Es handelt sich um die von Python empfohlene Methode für die Arbeit mit MongoDB. Die Verwendung ist sehr simpel und intuitiv.

Für die Installation wird der Package-Installer pip für Python empfohlen. Mit folgendem Command lässt sich PyMongo auf jedem beliebigen System, auf dem pip installiert ist, installieren. Im Anschluss muss die Library importiert werden. (vgl. MongoDB o. J. b)

```
python -m pip install pymongo
```

Listing 44. CMD-Shell: PyMongo Installation

```
from pymongo import MongoClient
```

Listing 45. Python-Code: PyMongo importieren

8.1.3 Datenbankplanung

Die Planung für die Echtdaten-Datenbank enthält sowohl das Planen der Struktur der einzelnen Dokumente als auch das Festlegen der Schnittstellen. Die *JSON-Struktur* bestimmt, wie die Dokumente in der Datenbank gespeichert werden. Die Rohdaten müssen dem Datenbank-Skript mittels CSV-Dateien in einem vordefinierten Format übergeben werden. Dieses Format ist als jeweilige Schnittstelle definiert.

Die Änderung der Planung ist unausweichlich gewesen. Durch das Erfordernis einer Anpassung, da beispielsweise bestimmte Daten nicht geliefert werden konnten, ist dies entsprechend aktualisiert worden. Zum Beispiel wurde das Erarbeiten des Endzeitpunktes eines Portscans gestrichen. Es handelt sich hierbei um die finalen Gegebenheiten.

8.1.3.1 Datenbankstruktur

Die Dokumente werden pro Themenbereich gespeichert. Für diese sind sogenannte *Collections* vorgesehen. *Collections* kann man sich wie Listen von Dokumenten, welche demselben Bereich zugeordnet sind, vorstellen. Daraus ergeben sich die aus Abbildung 23 hervorgehenden *Collections*[7].

[7] Die vollständige Datenbankplanung ist als externe Datei „Specto_Datenstruktur_Konzept.pdf“ auf der CD des Bibliotheksexemplars hinterlegt.

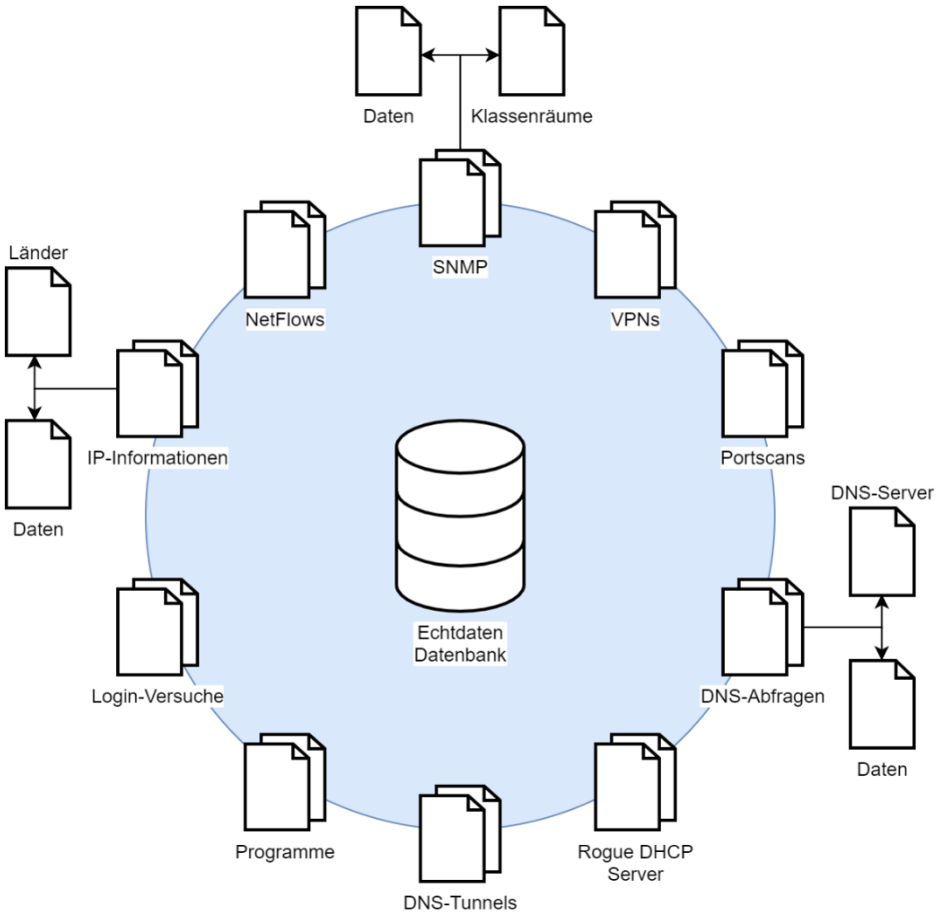


Abbildung 23. Datenbankstruktur

Speichern von Zeitpunkten

Heutzutage ist es schwer, Zeitangaben allgemein verständlich zu halten. Unterschiedliche Zeitzonen sowie kulturelle Formatierungen erschweren dies gewaltig. Daher wird bei der Speicherung von Daten und Uhrzeiten in dieser Diplomarbeit auf den *Standard ISO 8601* gesetzt. (vgl. IONOS 2020)

Sowohl Python als auch MongoDB können mit diesem Format sehr gut umgehen. In Python bietet die Library **`datetime`** eine Klasse mit diversen praktischen Methoden. Von MongoDB wird dieser Datentyp von vornherein unterstützt. Dadurch bietet sich dieses Format an und schafft die perfekten Gegebenheiten für eine übergreifende Verwendung.

Speichern von IP-Adressen

Das Speichern einer IPv4-Adresse als String benötigt mindestens 16 Byte. Eine IPv4-Adresse besteht jedoch nur aus 4 Byte (32 Bit). Daher ist es nahe liegend, IP-Adressen in der Datenbank als Integer abzuspeichern.

Feld für Zustand der Statistiken-Verarbeitung

Sämtliche für die Statistiken relevante Datensätze müssen den Zustand ihrer Verarbeitung speichern. Hierfür gibt es das Feld **`processed`**. Dieses speichert, ob der Datensatz bereits in den Statistiken vorhanden ist oder nicht.

Weiters existiert das Feld **`selected`**. Dieses gibt an, ob ein Datensatz im derzeitigen Durchlauf der Statistiken-Erstellung bereits miteinberechnet wird. Am Anfang eines Durchlaufes wird das Feld bei den derzeit vorhandenen Datensätzen auf **`True`** gesetzt. Auf diese Weise wird vermieden, dass Datensätze fälschlicherweise als **`processed`** markiert werden, sofern diese mitten im Durchlauf hinzugefügt werden. Es handelt sich dabei so gesehen um einen Sicherheitsmechanismus.

SNMP-Datensätze

Die SNMP-Daten werden pro Klasse gespeichert. Hierfür gibt es zuvor angelegte Dokumente für sämtliche Räume, deren Daten von „Nexus“ erfasst werden können. Die tatsächlichen Einträge verweisen anschließend auf die entsprechenden Klassenräume.

Die ID eines Klassenraums entspricht der Raumnummer einer Klasse. Weiters werden eine Beschreibung – beispielsweise die Stammklasse oder die Raumbezeichnung – sowie dazugehörige SNMP-Sensoren – zum Beispiel ein Sensor für WLAN- und ein weiterer für LAN-Netzwerkverkehr – gespeichert. Ein Dokument sieht also beispielsweise wie folgt aus:

```json
{
   "_id": 277,
   "description": "5AX",
   "sensors": [2040, 2041]
}
```

Listing 46. JSON: Klassenraum

Für einen SNMP-Eintrag wird die ID zufällig generiert. Die quantitative Datenmenge wird in Byte gespeichert. Weiters wird ein Zeitstempel, von wann der Eintrag stammt, sowie die ID der entsprechenden Klasse gespeichert. Ein Dokument kann folgendermaßen aussehen:

```json
{
   "_id": { "$oid": "5f9aefc1f2df65907359cca0" },
   "traffic": 6232.9207,
   "timestamp": { "$date": "2020-08-11T17:58:07.000Z" },
   "classroom_id": 277,
   "processed": false,
   "selected": false
}
```

Listing 47. JSON: SNMP

VPN- und Programm-Datensätze

VPN-Daten und Programm-Daten werden sehr ähnlich gespeichert. Für beide Teilbereiche gibt es eine *Collection*, welche sämtliche Verbindungen beinhält. Für den Fall, dass eine gleiche Verbindung bereits besteht, wird die vorherige aktualisiert.

Beide erhalten eine zufallsgenerierte und einzigartige ID. Sowohl der Ausgangspunkt sowie das Ziel der Verbindung wird gespeichert. Eine VPN-Verbindung kann sowohl einem Protokoll als auch einer Kategorie zugeordnet werden. Dafür kann eine Programm-Verbindung einem gewissen Programm zugeordnet werden. Weiters wird für beide Bereiche die Anzahl der übertragenen Pakete sowie die Summe der Daten festgehalten. Anhand von Start- und Endzeitpunkt können die Verbindungen einem Darstellungszeitraum zugeordnet werden. Außerdem muss gespeichert werden, ob eine Verbindung bereits beendet ist oder noch aktualisiert werden kann.

DNS-Datensätze

Weiters werden die externen DNS-Abfragen auf bestimmte DNS-Server gespeichert. Für die DNS-Server werden dynamisch neue Dokumente erstellt, beziehungsweise bei Bedarf aktualisiert. Die tatsächlichen Einträge verweisen auf den entsprechenden DNS-Server und beinhalten die abgefragte *Domain*.

Die ID eines DNS-Servers ist dessen öffentliche IP-Adresse. Für jeden DNS-Server wird gespeichert, welche *Domains* von diesem abgefragt werden. Dies ergibt eine Liste von mehreren *Domains* innerhalb eines DNS-Servers. Eine *Domain* ist über die tatsächliche abrufbare *FQDN*[8] identifizierbar. Weiters beinhält diese sämtlich damit verbundenen Domain-IP-Adressen in einer weiteren Liste. Ein DNS-Server-Eintrag hat somit folgende Struktur:

```
{
   "_id": 221186817,
   "domains": [{
         "domain": "www.illegaleSeite.com",
         "domainips": [ 3132370177, 3132370178 ]
   },
   {
         "domain": "www.illegaleSeite.at",
         "domainips": [3366308065]
   }]
}
```

Listing 48. JSON: DNS-Server

Die tatsächliche DNS-Abfrage wird in einer eigenen *Collection* gespeichert und erhält eine zufällige ID. Die Source-IP-Adresse gibt an, von welchem Host die Abfrage ausgeht. Die Destination-IP-Adresse verweist auf einen DNS-Server. Durch die *Domain* kann identifiziert werden, welche *Domain* des DNS-Servers tatsächlich abgefragt wird und welche IP-Adressen dazugehören. Ein Eintrag kann wie folgt aussehen:

```
{
   "_id": { "$oid": "5fdf269991a9f677f480cde6" },
   "sourceip": 167772161,
```

[8] Fully Qualified Domain Name: vollständige und eindeutige Adresse

```
    "destinationip": 134744072,
    "timestamp": { "$date": "2020-12-18T12:00:00.000Z" },
    "domain": "www.illegaleSeite.com",
    "processed": false,
    "selected": false
}
```

Listing 49. JSON: DNS-Abfrage

Sonstige Datensätze

Ebenso gibt es eine jeweilige *Collection* für Portscans, NetFlow-Verbindungen, DNS-Tunnels und Rogue-DHCP-Server.

Für jeden Eintrag wird eine zufällige ID generiert. Weiters werden die Source- und Destination-IP-Adresse gespeichert. Für die Rogue-DHCP-Server-Einträge wird die Adresse Source angesehen. Sämtliche Einträge besitzen einen Startzeitpunkt, um sie einem Zeitraum zuordnen zu können. NetFlow-Verbindungen werden weiters einem Protokoll sowie einer Kategorie zugeordnet. Außerdem wird die Summe der über einen DNS-Tunnel übertragenen Daten festgehalten.

IP-Datensätze

Für sämtliche Dokumente, welche IP-Adressen beinhalten, werden die Einträge für IP-Adressen benötigt. Die eigentlichen Dokumente für IP-Adressen beinhalten verschiedene Informationen zu der jeweiligen IP-Adresse und verweisen auf das Land, aus welchem diese Adresse stammt.

Für die Darstellung der verschiedenen Verbindungen auf der Weltkarte werden entsprechende Koordinaten benötigt. Jedes Land hat als ID den entsprechenden Länder-Code. Durch eine von Google bereitgestellte Liste von Ländern mit ihren Koordinaten (Google Developers o. J.) können anschließend für sämtliche Länder der Breiten- und Längengrad gefüllt werden. Außerdem wird der Name des Landes gespeichert. Die Struktur für ein entsprechendes Land sieht wie folgt aus:

```
{
    "_id": "AT",
    "latitude": 47.516231,
    "longitude": 14.550072,
    "name": "Austria"
}
```

Listing 50. JSON: Land

Sämtliche erkannten IP-Adressen werden als Dokument gespeichert. Es wird die IP-Adresse selbst, eine dazugehörige Domain, die AS-Nummer sowie der entsprechende Ländercode gespeichert. Der Ländercode verweist auf das entsprechende Land mit den Koordinaten. Ein Dokument sieht wie folgt aus:

```
{
    "_id": { "$oid": "5fa5aed3690e4708646eddee" },
    "address": 3567680042,
    "source": 167772161,
    "domain": "www.magenta.at",
    "asn": 100,
    "country": "AT",
    "timestamp": { "$date": "2020-12-18T12:00:00.000Z" },
    "processed": false,
    "selected": false
}
```

Listing 51. JSON: IP

8.1.3.2 Datenbankschnittstellen

Das Skript für das Speichern sämtlicher Datensätze setzt das Angeben der Dateien, deren Einträge gespeichert werden sollen, voraus. Hierfür ist ein zentralisierter Ordner bereitgestellt. Sämtliche, in darunter liegenden Unterordnern gespeicherte Dateien, werden automatisch mittels Cronjobs eingelesen (siehe Kapitel 9.2).

Für die jeweils gespeicherten Dateien gibt es ein vordefiniertes Format, in welchem die CSV-Dateien gespeichert werden müssen. Wird das jeweilige Format nicht eingehalten, können die darin enthaltenen Datensätze nicht in der Datenbank gespeichert werden. Da sämtliche gewonnene Daten ohnehin automatisch ausgewertet werden, kann es hierbei zu keinen Fehlern kommen. Pro Zeile ist ein Eintrag enthalten. SNMP-Daten[9] müssen im folgenden Format vorhanden sein:

```
<Datenmenge>;<Zeitpunkt>;<Klassenraum-ID>
```

Listing 52. CSV: SNMP-Aufbau

Es kann somit folgende Datei erhalten und im Anschluss eingelesen werden:

```
6232.9207;15.01.2021 12:25:00;2040
10481.5304;15.01.2021 12:25:00;2041
8132.901;15.01.2021 12:25:00;2042
5412.65;15.01.2021 12:25:00;2043
```

Listing 53. CSV: SNMP-Beispiel

8.1.4 Umsetzung des Skripts

Das Skript[10] ist in Python programmiert und verwendet die Library PyMongo zur Speicherung der Datensätze, wie oben bereits erwähnt wurde. Um diese verwenden zu können, wird aus der Library die Klasse **`MongoClient`** importiert. Dies geschieht mit der Code-Zeile aus Listing 54.

[9] Die vollständige Übersicht der Schnittstellen ist als externe Datei „Specto_Datenstruktur_Schnittstellen.pdf“ auf der CD des Bibliotheksexemplars hinterlegt.

[10] Das vollständige Skript ist als externe Datei „MongoDB.py“ auf der CD des Bibliotheksexemplars hinterlegt.

```
from pymongo import MongoClient
```

Listing 54. Python-Code: MongoClient importieren

Anschließend kann ein entsprechendes Objekt initiiert werden, womit die Methoden zum Speichern und Auslesen der Datensätze verwendet werden können.

Zur Herstellung der Verbindung wurde die Methode **`initiate_connection()`** erstellt. Diese speichert die Verbindung in der Variable **`client_specto`**, welche für die späteren Datenbankzugriffe verwendet wird. Für den Aufbau der Verbindung wird ein Connection-String benötigt. Der Aufbau sieht wie folgt aus:

```
mongodb://<username>:<password>@<ip-address>:<port>/
```

Listing 55. Aufbau: Connection-String

Dieser Connection-String wird dem Konstruktor der **`MongoClient`**-Klasse übergeben, wie in Listing 56 zu sehen ist. Die hierfür benötigten Parameter sind abhängig von der Konfiguration des DMZ-Hosts im Schulnetzwerk.

```
client_specto = MongoClient(<connection-string>)
```

Listing 56. Python-Code: MongoClient Definition

Die daraus erstellte Variable wird in den Methoden zur Speicherung der Daten weiterverwendet. Jede dieser Methoden kann im Anschluss mittels **`argparse`** einzeln über das Skript aufgerufen werden.

Speicherung von IP-Adressen

Um eine IP-Adresse in einen Integer-Wert umrechnen zu können, wurde die Methode **`ip_to_int()`** programmiert. Diese ermöglicht es unter Angabe der IP-Adresse als String diese in einen Integer-Wert umzurechnen.

Für die Umwandlung verwendet die Methode die **`ipaddress`**-Library. Diese ermöglicht es, eine IP-Adresse als Objekt zu speichern und bietet anschließend diverse Methoden. So kann man sich beispielsweise das Objekt als Integer zurückliefern lassen.

Die Methode versucht also, aus dem übergebenen String ein **`IPv4Address`**-Objekt zu erstellen. Dies wird mittels der folgenden Zeilen bewerkstelligt:

```
try:
    ip_address = ipaddress.IPv4Address(string_ip_address)
except AddressValueError:
    …
```

Listing 57. Python-Code: IPv4Address-Objekt erstellen

Sofern dies jedoch nicht möglich ist, wird die Exception **`AddressValueError`** erwartet. Tritt diese ein, wird versucht, ein IPv6Address-Objekt daraus zu bilden. Sofern das auch nicht möglich ist, handelt es sich um keine gültige IP-Adresse und es wird **`0`** zurückgeliefert. Kann die Methode ohne Fehler ausgeführt werden, wird die IP-Adresse als Integer zurückgeliefert.

```
return int(ip_address)
```

Listing 58. Python-Code: Zurückliefern der IP-Adresse als Integer

Speichern von SNMP-Datensätzen

Die Methode **`save_snmp_entries(file)`** dient dem Speichern von SNMP-Einträgen. Die übergebene CSV-Datei enthält gemäß der Datenbankschnittstellen einen Eintrag pro Zeile. Die einzelnen Werte eines Eintrags werden in einer Liste gespeichert. Eine Klasse kann mehrere Sensoren besitzen, wobei die von diesen stammenden Datenmengen in einem einzigen Datensatz zusammengefasst gespeichert werden.

Zuerst werden die Datenbank und die *Collection* **`classrooms`** als Variablen gespeichert. Daraufhin werden die in der *Collection* **`classrooms`** gespeicherten Klassenräume nach dem übergebenen Sensor durchsucht. Hierfür wird eine entsprechende *Query* angelegt und diese weiter für das Durchsuchen verwendet. Der entsprechende Raum wird in einer Variable gespeichert und überprüft, ob dieser nicht **`None`** ist.

```
db_specto = client_specto["specto"]
col_classrooms = db_specto["classrooms"]

query = {"sensors": int(sensor)}
possible_classroom = col_classrooms.find_one(query)
```

Listing 59. Python-Code: Collection classrooms durchsuchen

In einer Liste werden die bereits durchgegangenen Klassenräume gespeichert. Ist ein Klassenraum in dieser Liste noch nicht enthalten, werden die gewonnenen Informationen in einem *Dictionary* gespeichert und anschließend der *Collection* **`snmpentries`** zum Speichern übergeben. Diese *Collection* wird hierfür zuvor in einer Variable gespeichert.

```
col_snmp_entries = db_specto["snmpentries"]
col_snmp_entries.insert_one(ready_snmp_entry)
```

Listing 60. Python-Code: SNMP-Eintrag speichern

Sofern für den Klassenraum in diesem Durchlauf bereits ein Datensatz besteht, wird das zuvor erstellte Dokument lediglich aktualisiert.

Der zuvor in Worten beschriebene Ablauf lässt sich am besten mithilfe des folgenden Ablaufdiagramms veranschaulichen:

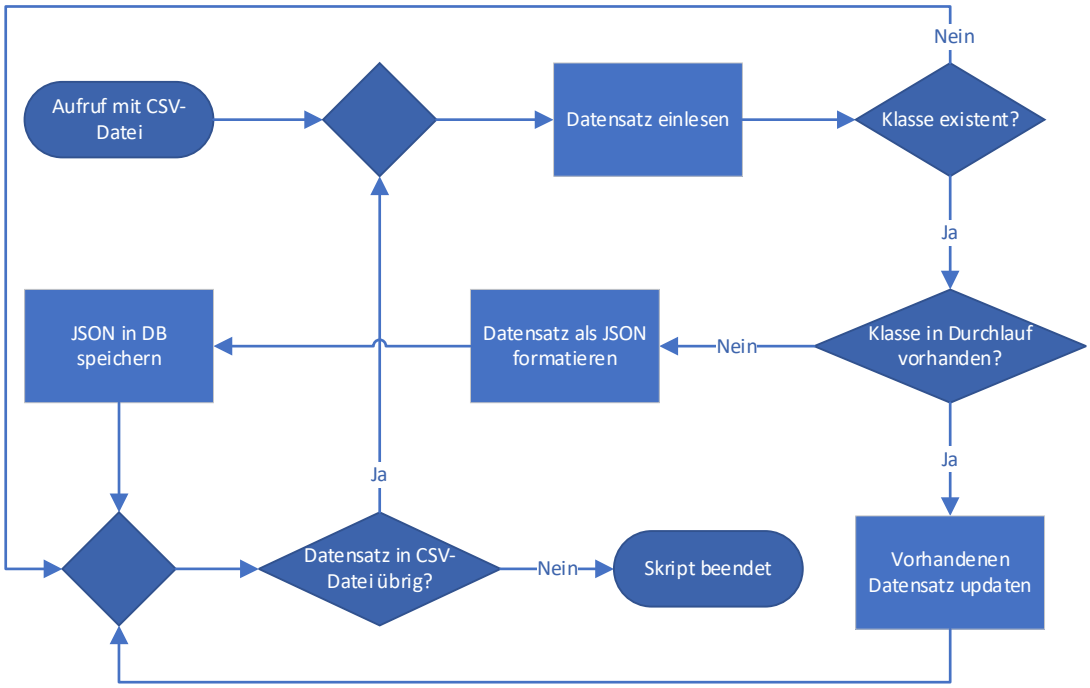


Abbildung 24. Ablauf: Verarbeitung der SNMP-Datensätze

Speichern von DNS-Datensätzen

Die Methode **`save_dnsexternal_entries(file)`** wird für das Speichern von externen DNS-Abfragen verwendet. Beim Speichern dieser Einträge kommt zusätzlich das Instandhalten der gespeicherten DNS-Server hinzu.

Das Speichern der tatsächlichen Daten in der *Collection* **`dnsexternalentries`** ist vom Prinzip her dasselbe wie bei den SNMP-Einträgen.

Zusätzlich muss jedoch überprüft werden, ob der DNS-Server bereits in der *Collection* **`dnsserver`** vorhanden ist. Dies wird mittels einer *Query* bewerkstelligt. Ist dieser vorhanden, wird geprüft, ob die in der Abfrage aufgelöste *Domain* bereits in diesem DNS-Server-Eintrag enthalten ist. Mittels einer *Query* wird auf Vorhandensein kontrolliert.

```
domain_query = {"_id": dst_ip, "domains.domain": domain_name}
dnsserver_domain = col_dnsservers.find_one(domain_query)
```

Listing 61. Python-Code: DNS-Server auf Domain prüfen

Anschließend wird entweder die *Domain* mit den neu aufgelösten IP-Adressen geupdatet oder der DNS-Server mit der neuen *Domain* geupdatet. Die aufgelösten IP-Adressen werden zuvor aus der CSV-Datei ausgelesen und in der Liste **`domain_ips`** gespeichert.

```
if dnsserver_domain is not None:
   col_dnsservers.update_one(domain_query, {'$set':
        {'domains.$.domainips': domain_ips}})
```

```
else:
   col_dnsservers.update_one(query, {'$push': {'domains': {"domain":
         domain_name, "domainips": domain_ips}}})
```

Listing 62. Python-Code: Domain hinzufügen/updaten

Der Ablauf des Speicherns der tatsächlichen DNS-Abfragen ist in Abbildung 27 ersichtlich. Neben diesem Ablauf, findet auch der Ablauf der Erstellung beziehungsweise des Aktualisierens der DNS-Server innerhalb der Echtdaten-Datenbank statt:

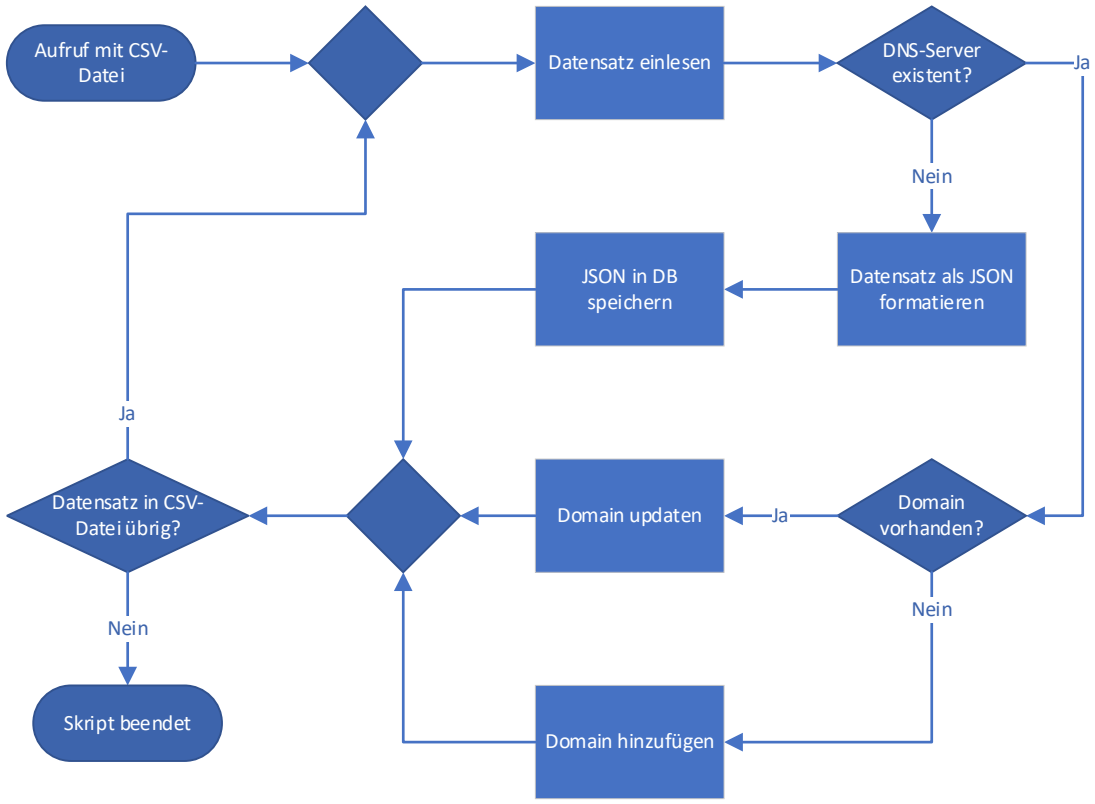


Abbildung 25. Ablauf: Verwaltung DNS-Server

Speichern von VPN- und Programm-Datensätzen

Beim Speichern von VPN- und Programm-Einträgen muss zuvor überprüft werden, ob es eine Art derselben Verbindung gibt, welche noch nicht beendet ist. Hierfür gibt es die Methode **`save_vpn_entries(file)`**.

Es wird zuerst mittels *Query* die *Collection* **`vpnentries`** beziehungsweise **`programentries`** durchsucht, ob eine Verbindung mit denselben Werten (**`Source`**, **`Destination`** & **`type`**) existiert, welche noch nicht abgeschlossen ist (**`isfinished`** gleich **`False`**).

```
filter_vpn = {"sourceip": src_ip, "destinationip": dst_ip,
   "sourceport": int(src_port), "destinationport": int(dst_port),
   "isfinished": False, "type": type}
```

Listing 63. Python-Code: VPN-Query

Ergibt diese Suche einen Treffer, so wird bei diesem Eintrag der Endzeitpunkt, der Verbindungsstatus, die Anzahl der Pakete und die Verbindungsgröße aktualisiert. Für die beiden Werte, die neu gesetzt werden müssen, wird der *Feldoperator* **`$set`** und für jene, deren Werte erhöht werden müssen, wird der **`$inc`** *Operator* verwendet. Diese *Operatoren* werden mit den neuen Werten in einem *Dictionary* gespeichert und der Update-Methode gemeinsam mit einer *Query* übergeben.

```
if col_vpn_entries.find_one(filter_vpn) is not None:
   new_values = {"$set": {"stop": timestamp, "isfinished":
                bool(int(is_finished))},
         "$inc": {"packetcount": int(packet_count), "connectionsize":
                float(connection_size)}}
   col_vpn_entries.update_one(filter_vpn, new_values)
```

Listing 64. Python-Code: VPN-Eintrag updaten

Ist keine derartige laufende Verbindung offen, wird ein neuer Eintrag erstellt und anschließend in der Datenbank gespeichert. Das folgende Ablaufdiagramm visualisiert den oben geschilderten Prozess:

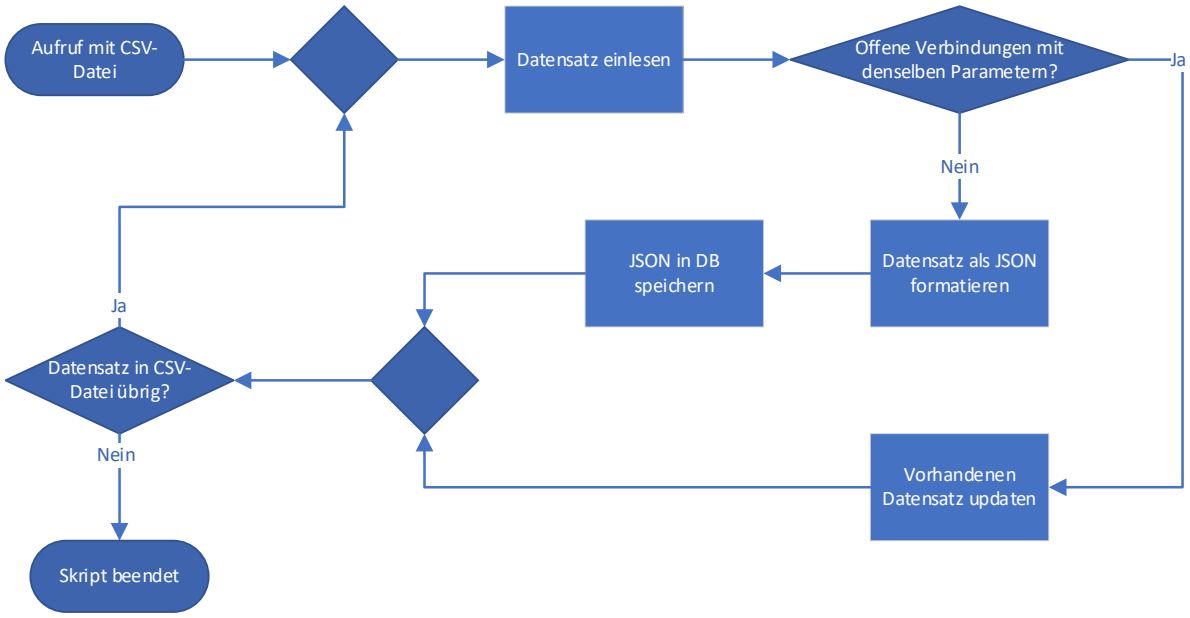


Abbildung 26. Ablauf: Verarbeitung VPN- und Programm-Datensätze

Speichern sonstiger Datensätze

Für sämtliche andere Datensätze sind ebenfalls Methoden implementiert. Diese ähneln dem Aufbau der SNMP-Methode. Es wird aus dem CSV-File lediglich ein Dictionary erstellt, welches anschließend der entsprechenden *Collection* übergeben wird.

Der Ablauf der Methoden ähnelt denen der anderen Datensätze, wobei bestimmte Überprüfungen ausgelassen werden können. Diese lassen sich mittels folgendem Ablaufdiagramm beschreiben:

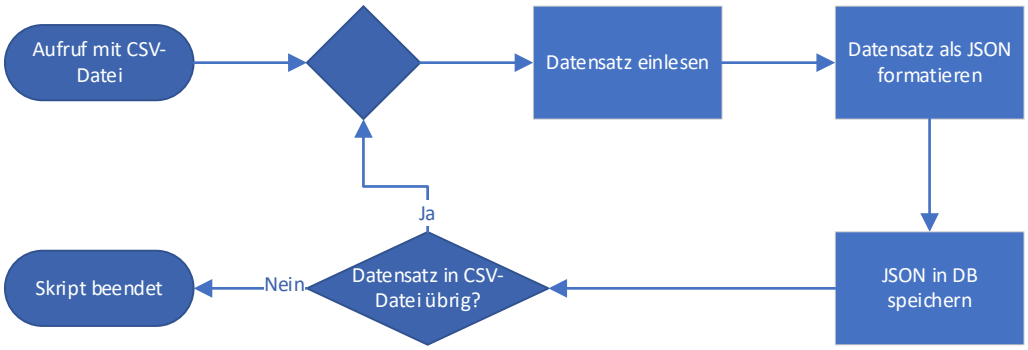


Abbildung 27. Ablauf: Verarbeitung sonstiger Datensätze

Timeout bei Verbindung

Für den Fall, dass es beim Aufbau einer Verbindung zum *Timeout* kommt, wird aus der Library der **`ServerSelectionTimeoutError`** importiert. Kommt es zum Verbindungsfehler, wird dieser entsprechend bei jeder Methode gecatcht und in den Logs vermerkt. Die einzulesende Datei wird dadurch nicht gelöscht.

```
from pymongo.errors import ServerSelectionTimeoutError
```

Listing 65. Python-Code: MongoDB Timeout-Error importieren

Weiters werden in den Logs neue Verbindungsversuche durch das Skript sowie das Speichern von neuen Datensätzen mitprotokolliert.

Argparse

Um das Skript über die CMD-Shell ausführbar zu machen, ist für sämtliche Methoden **`argparse`** implementiert. Hierfür muss zuerst die Library **`argparse`** importiert werden. Anschließend kann somit ein **`ArgumentParser`** erstellt werden. Dieser erhält zum einem das Verbose-Argument, welches den Benutzer zusätzlichen Output des Skripts sehen lässt.

Zusätzlich gibt es zwei Sub-Parser, **`save`** und **`read`**, welche einerseits zum Beschreiben und andererseits zum Auslesen der Datenbank dienen. Diese erhalten eine Gruppe von Argumenten, welche dafür sorgen, dass man mit jedem Mal ausführen eine Art von Einträgen beispielsweise speichern lassen kann. Mit folgendem Befehl kann man beispielsweise die SNMP-Einträge aus dem File **`snmp.csv`** einlesen:

```
./MongoDB.py save -snmp snmp.csv
```

Listing 66. CMD-Shell: Speichern von SNMP-Einträgen

8.1.5 Beispieldatensätze

Sämtliche für die Diplomarbeit relevanten Datensätze können nun in der Echtdaten-Datenbank gespeichert werden. Die folgenden Selects beziehen sich auf die Mongo-Shell find-Methode und nicht die von der PyMongo-Library. Ein Select kann wie folgt aufgebaut sein:

```
db.collection.find(query, projection)
```

Listing 67. Mongo-Shell: Select-Aufbau

Mithilfe dieses Selects wird ein *Cursor*, welcher auf die ausgewählten Dokumente referenziert, zurückgeliefert. Die Parameter sind laut der MongoDB Dokumentation wie folgt anzuwenden. (vgl. MongoDB o. J. c)

Tabelle 2. MongoDB find() Parameter (vgl. MongoDB o. J. c)

Parameter	Beschreibung
Query	Der Parameter ist optional und kann den Filter mithilfe von Abfrageoperatoren angeben. Wird der Parameter weggelassen oder ein leeres Dokument übergeben, werden alle Dokumente der *Collection* zurückgegeben.
Projection	Der Parameter ist optional und gibt an, welche Felder der Dokumente, welche dem Filter entsprechen, zurückgeliefert werden sollen. Wird der Parameter leer gelassen, werden alle Felder zurückgegeben.

Eine für die Medientechnik benötigte typische Abfrage wäre für SNMP die Abfrage des letzten Datenmengeneintrages einer bestimmten Klasse. Hierfür wird in diesem Fall über die Shell zuerst die benötigte Sensor-ID abgefragt und in einer Variablen (**`classroom_id`**) abgespeichert werden. Diese Variable kann schließlich in der *Query* dazu verwendet werden, um die Einträge auf eine Klasse zu beschränken.

Die Zwischenschritte können mittels Python nochmals viel einfacher gestaltet werden.

```
> var classroom = (db.classrooms.findOne({"nr": 277}))._id

> db.snmpentries.findOne({$query: {"timestamp": { $gt: new Date('2021-01-
15')}, "classroom_id": classroom}, $orderby: {"timestamp": -1}}, {"_id":
0, "traffic": 1})
```

Listing 68. Mongo-Shell: SNMP-Select

Dieser Select liefert folgende Ausgabe:

```
{ traffic: 5130.72 }
```

Listing 69. Mongo-Shell: SNMP-Select Ausgabe

Um VPN-Verbindungen darstellen zu können, werden neben den Informationen des tatsächlichen Eintrags auch die Koordinaten benötigt. Mittels Python kann somit der VPN-Eintrag in einer Variable gespeichert werden, um anschließend auf die Destination zugreifen zu können und das dazugehörige Land herauszufinden. Durch den Ländercode gelangt man anschließend an die Koordinaten. Ein Select für das letzte aufgebaute VPN könnte wie folgt aussehen:

```
> var vpn = db.vpnentries.findOne({$query: {"stop": { $lt: new
Date('2021-01-10T12:00:00Z')}}, $orderby: {"stop": -1}})

> var ip = db.ipentries.findOne({$query: {"address": vpn.destinationip}})
```

```
> var country = db.countries.findOne({$query: {"_id": ip.country}})

> printjson({'source': vpn.sourceip, 'destination': vpn.destinationip,
'latitude': country.latitude, 'longitude': country.longitude})
```

Listing 70. Mongo-Shell: VPN-Select

Die Ausgabe kann somit auf das Mindeste und auf ein einzelnes JSON-Dokument reduziert werden. Folgendes kann dann anschließend in einer ähnlichen Form von mehreren Einträgen von der API an die Medientechnik zur Verfügung gestellt werden.

```
{
  source: '10.0.0.1',
  destination: '8.8.8.8',
  latitude: 37.09024,
  longitude: -95.712891
}
```

Listing 71. Mongo-Shell: VPN-Select Ausgabe

8.2 Statistik-Datenbank

Die Statistik-Datenbank fasst sämtliche in der Diplomarbeit erstellte Statistiken zusammen. Diese werden anschließend dazu verwendet, um sie für die Erstellung diverser grafischer Darstellungen an die Medientechnik bereitzustellen.

Die Statistiken werden in einer *relationalen Datenbank* zentralisiert gespeichert. Dies ermöglicht es anschließend der API schnelle Abfragen für die Medientechnik tätigen zu können. Die Werte müssen nicht erst für jede Abfrage berechnet werden, sondern stehen bereits fertig zur Verfügung.

8.2.1 Relationale Datenbanken und MySQL

Spricht man von einer *relationalen Datenbank*, meint man eine Datenbank, welche aus mehreren Relationen (Tabellen) besteht. In diesen Tabellen werden logisch zusammengehörende Daten gespeichert. Die *Structered Query Language* (SQL) hat sich dabei als Standard der *Datenmanipulationssprachen* durchgesetzt. (vgl. Fuchs 2018: 44)

Generell kann man sagen:

> Relationale Datenbanken ist das am weitesten verbreitete Datenbankmodell. Es setzt auf das relationale Datenbankmodell, das auf der Speicherung von Informationen in verschiedenen Tabellen basiert, die untereinander über Beziehungen (Relationen) verknüpft sind. (Luber 2017b)

Bei MySQL handelt es sich um ein von Oracle entwickeltes relationales *Open-Source-SQL-Datenbankverwaltungssystem*. Es ist eines der weltweit am häufigsten verwendeten *Datenbanksysteme*. Große Datenmengen können schnell und performant verarbeitet werden. (vgl. Luber 2017c)

8.2.2 MySQL mit Python

Die MySQL-Connector-Library ermöglicht ein nahtloses Zusammenarbeiten zwischen einer MySQL-Datenbank und einem Python-Skript. Fast sämtliche und alle für die Diplomarbeit relevanten Features von MySQL-Server 8.0 werden mit der allbekannten *Syntax* unterstützt. Ebenso ist das Konvertieren von Parameterwerten zwischen Python- und MySQL-Datentypen möglich. (vgl. MySQL o. J.)

Für die Installation kann auch hier der Package-Installer pip verwendet werden. Mittels folgender Zeile kann die oben erwähnte Library installiert werden.

```
python -m pip install mysql-connector-python
```

Listing 72. CMD-Shell: Installation des MySQL-Connectors

Im Gegensatz zum abweichenden Namen bei der Installation, muss diese mit folgender Codezeile im Programmcode importiert werden:

```
import mysql.connector
```

Listing 73. Python-Code: Importieren des Connectors

8.2.3 Datenbankplanung

Eine beispielhafte Fragestellung für von der Diplomarbeit gespeicherte Statistiken wäre etwa: „Wie viele Daten wurden von der IP-Adresse „10.0.79.21“ (1. Dimension) über VPNs (2. Dimension) im Jahr 2020 (3. Dimension) produziert?“ Hierfür werden die drei Dimensionen Source, Verbindung und Zeit benötigt und abgefragt. (vgl. Manhart 2008a)

Dadurch ergibt sich das Erfordernis für die Betrachtung eines *multidimensionalen Datenraumes*. Dieser lässt sich mittels eines *Star-Schemas* darstellen.

Die Statistik-Werte sind insbesondere für den Netzwerkadministrator interessant. Aufgrund dessen wurden diese gemeinsam mit dem Auftraggeber auf die wichtigsten Kennwerte reduziert[11].

8.2.3.1 Star-Schemen

Das *Star-Schema* ist eine Form des *multidimensionalen Datenmodells*. Das zum Standard für die Darstellung derartiger Modelle entwickelte Schema wird in *Data Warehouses* und *OLAP-Anwendungen* (siehe hierfür Kapitel 1.2.3.2 unten) eingesetzt. Die Bezeichnung stammt von der sternenförmigen Anordnung der Dimensionen (Tabellen). (vgl. Manhart 2008a)

Im Fall von drei Dimensionen bildet die Datenstruktur einen Würfel. Bei mehr als drei *Dimensionstabellen* ist die Rede von einem Hyperwürfel. (vgl. Nguyen 2018)

[11] Die vollständige Datenbankplanung ist als externe Datei „Specto_Datenstruktur_Konzept.pdf“ auf der CD des Bibliotheksexemplars hinterlegt.

Ein grundlegender Aufbau sieht wie folgt aus:

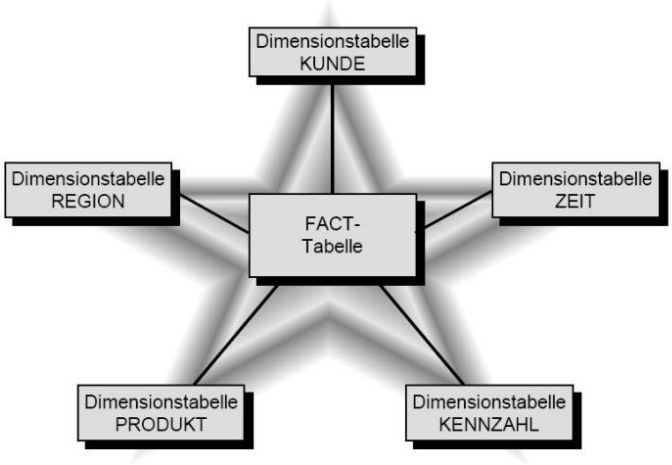


Abbildung 28. Star-Schema.
Quelle: https://images.tecchannel.de/bdb/364608/840x473.webp

Anhand der Abbildung lässt sich erkennen, dass sich die Daten eines *Star-Schemas* in zwei Gruppen unterteilen lassen:

- *Faktentabelle*: Diese Tabelle steht im Zentrum. In dieser werden die zu verwaltenden oder zu analysierenden Daten gespeichert. Stellt man sich die Struktur als Würfel vor, handelt es sich hierbei um den Kern. Die als *Fremdschlüssel* gespeicherten *Primärschlüssel* der *Dimensionstabellen* bilden zusammen den *Primärschlüssel* der Fakten. (vgl. Manhart 2008a)
- *Dimensionstabellen*: Diese Tabellen beschreiben die Fakten mit dazugehörigen Attributen beziehungsweise Eigenschaften. Jeder Datensatz einer Dimension wird durch einen *Primärschlüssel* identifiziert. (vgl. Nguyen 2018)

Um die redundante Speicherung von Werten zu verhindern, werden Tabellen normalisiert. Dabei wird angestrebt, Redundanzen auf ein Minimum zu reduzieren. Eine höhere *Normalform* hat jedoch die Abnahme der Abfrage-Performance zur Folge. (vgl. Nguyen 2018)

Ziel des *Star-Schemas* ist somit nicht die Normalisierung der Tabellen, sondern *Star-Schemen* sind darauf ausgelegt, Leseoperationen möglichst effizient durchführen zu können. *Dimensionen* sind bis zur zweiten *Normalform* und *Faktentabellen* bis zur dritten *Normalform* optimiert. Daraus folgt eine geringe Anzahl an Join-Operationen, um an benötigte Daten zu kommen. (vgl. Manhart 2008a; Nguyen 2018; Datenbanken-verstehen o. J.)

Klassenbezogenes Star-Schema

Die klassenbezogenen Statistiken befassen sich lediglich mit SNMP-Datensätzen. Daraus lässt sich ableiten, dass sich diese Werte mit den produzierten Daten einzelner Klassen beschäftigen. Eine typische Frage wäre: „Wie viele Daten wurden durchschnittlich vom Klassenraum „277“ (1. Dimension) im Jänner 2021 (2. Dimension) produziert?“

Daraus folgen zwei *Dimensionstabellen*:

- CLASS: In der Dimension werden die verschiedenen Klassenräume gespeichert. Zu den gespeicherten Feldern zählen die Raumnummer und eine Beschreibung. Der *Primärschlüssel* ist eine ID und befindet sich als *Fremdschlüssel* in der Faktentabelle.
- TIME: Diese Dimension speichert den Tag, das Monat und das Jahr. Die verschiedenen Kombinationen werden durch eine ID als *Primärschlüssel* eindeutig identifiziert. Diese ID wird in der *Faktentabelle* als *Fremdschlüssel* gespeichert.

Die Faktentabelle speichert die Summe sowie den Durchschnitt der quantitativen Datenmenge und die Anzahl der hierfür analysierten Datensätze. Die Anzahl wird benötigt, um sich folglich den neuen Durchschnitt berechnen zu können. Der *Primärschlüssel* der *Faktentabelle* setzt sich aus den *Fremdschlüsseln* der beiden *Dimensionstabellen* zusammen.

Aus den soeben beschriebenen Tabellen ergibt sich das folgende Star-Schema:

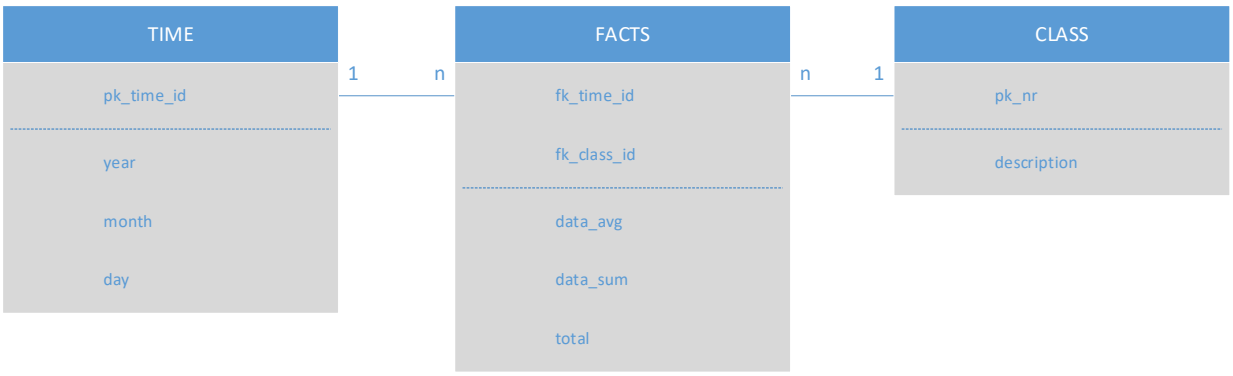


Abbildung 29. Klassenbezogenes Star-Schema

IP-bezogenes Star-Schema

Bei den IP-bezogenen Statistiken handelt es sich um Werte zu VPN-Verbindungen, Portscans, DNS-Abfragen und so weiter. Das hat zur Folge, dass die verschiedenen Verbindungen zu keiner Klasse, sondern zu einer Source und Destination zugeordnet werden können. Eine Frage könnte wie folgt lauten: Wie oft wurde von der IP-Adresse „10.76.0.17“ (1. Dimension) zur IP-Adresse „213.76.200.65“ (2. Dimension) eine VPN-Verbindung (3. Dimension) am 27. Jänner 2021 (4. Dimension) aufgebaut?

Aus dieser Fragestellung lassen sich die folgenden *Dimensionstabellen* ableiten:

- SOURCE: Die Dimension speichert die IP-Adresse, von welcher die Verbindung ausgeht, als *Primärschlüssel*. Diese wird als *Fremdschlüssel* in der *Faktentabelle* gespeichert.
- DESTINATION: Ähnlich zur SOURCE-*Dimensionstabelle* wird hier die IP-Adresse, die das Ziel der Verbindung ist, als *Primärschlüssel* gespeichert. Die *Faktentabelle* speichert die IP-Adresse als *Fremdschlüssel*.
- CONNECTION: Diese *Dimensionstabelle* speichert den Namen der Verbindung (z.B.: VPN), gegebenenfalls das benutzte Protokoll sowie den Typ. Die ID der Verbindung dient als *Primärschlüssel* und wird als *Fremdschlüssel* in der *Faktentabelle* abgespeichert.

- TIME: Diese Dimension ist ident zur TIME-*Dimensionstabelle* aus „Klassenbezogenes Star-Schema“.

In der *Faktentabelle* sind die Summe der übertragenen Daten sowie die Aufrufe dieser Verbindung enthalten. Weiters setzt sich der *Primärschlüssel* aus den *Fremdschlüsseln* der vier *Dimensionstabellen* zusammen.

Die beschriebenen Tabellen ergeben somit folgendes Star-Schema:

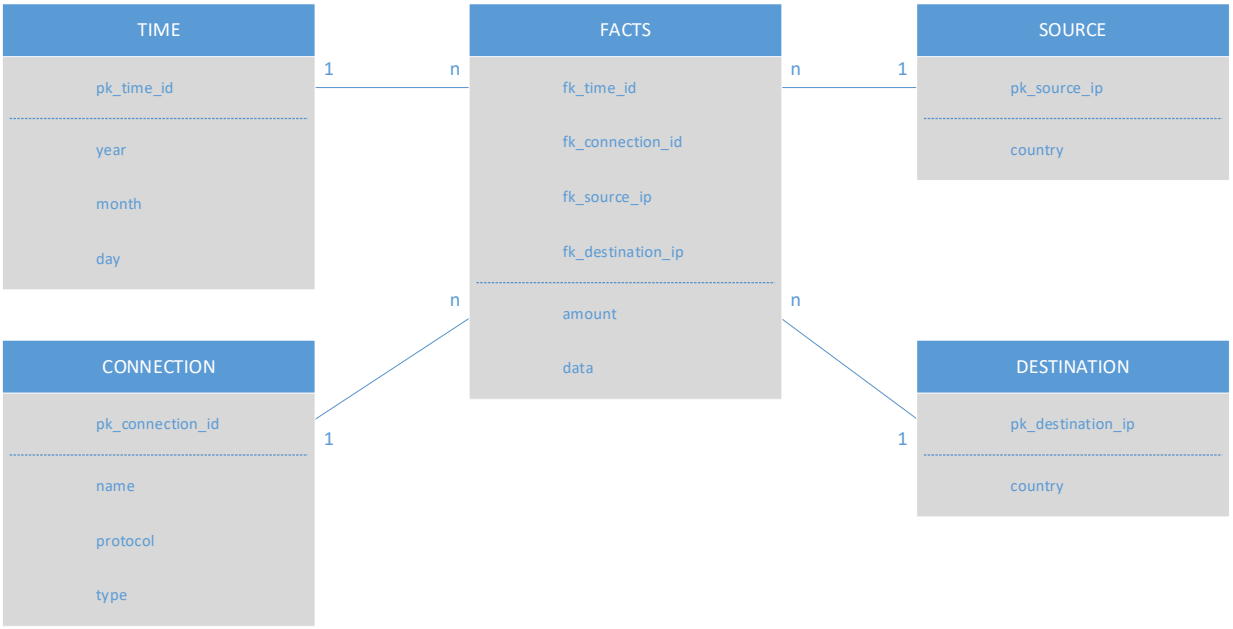


Abbildung 30. IP-bezogenes Star-Schema

8.2.3.2 OLAP-Cube im Vergleich

Die Speicherung von Daten in einem *OLAP-Cube* werden nicht wie bei *relationalen Datenmodellen* in flachen Tabellen, sondern *multidimensional* gespeichert. Dies ermöglicht die Betrachtung der Daten aus verschiedenen Blickwinkeln und Detaillierungsstufen. Derartige Kriterien sind beliebig kombinierbar. Bereiche des *Data-Warehousings* und des *Online Analytical Processings* (OLAP) sind Anwendungsfälle. (vgl. Manhart 2008b; Luber 2017d)

Dimensionen und Fakten sind die wesentlichen Komponenten eines *OLAP-Cube*. Die Dimensionen entsprechen dabei den Achsen des Würfels. Natürlich sind diese nicht auf drei Raumrichtungen begrenzt, sondern es kann beliebig viele geben. Bereits vordefinierte Grundoperationen wie *Slicing* (auf eine Dimension beschränken) oder *Dicing* (einen Teilwürfel herausschneiden) stehen zu Analysezwecken bereit. *Data Warehouses* sind im *Normalfall* unterhalb eines OLAP-Systems und dienen als Datenquelle. (vgl. Luber 2017d; Dörner 2019)

OLAP-Cubes basieren auf einem *Star-Schema*. Diese berechnen aggregierende und zusammenfassende Werte und verbessern somit die Abfrageleistung. Zur Analyse werden bereits zur Verfügung gestellte OLAP-Analysetools verwendet. Bei einem *OLAP-Cube* handelt es

sich also um ein bereits fertiges zur Verwendung bereitstehendes Produkt. (vgl. Rathish 2017)

Die untenliegende Abbildung veranschaulicht einen solchen *OLAP-Cube*. Die hervorgehobene Zelle gibt etwa den Umsatz des Produkts 1, in Region Ost im Jahr 2005 an:

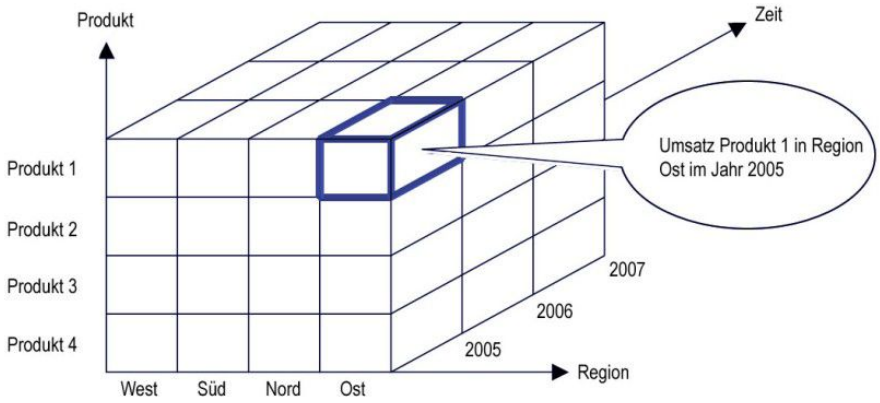


Abbildung 31. OLAP-Cube.
Quelle: https://images.tecchannel.de/bdb/362924/840x473.webp

8.2.3.3 Statistik-Werte

Bei den Statistik-Werten handelt es sich um jene Werte, welche in der jeweiligen *Faktentabelle* gespeichert werden. Diese sind je nach Bereich und Verbindungstyp unterschiedlich definiert worden. Sämtliche im folgenden Teil angeführten Informationen wurden im Vorhinein selbst erarbeitet und anschließend mit dem Auftraggeber fixiert. Dabei war es wichtig jene Werte, welche für den Netzwerkadministrator nicht von Relevanz sind, zu entfernen und jene, welche mitunter gefehlt haben, zu ergänzen oder abzuändern.

Zeiteinheiten

Grundsätzlich gilt, dass Statistiken für jede beliebige Zeiteinheit erstellt werden könnten. Nach Bedenken vom Netzwerkadministrator wurde festgelegt, dass für diese Diplomarbeit nur die Erstellung von täglichen, monatlichen und jährlichen Statistiken von Relevanz ist. Dementsprechend kommen die Felder von der TIME-*Dimensionstabelle* zustande.

IP-Kombinationen

Für IP-bezogene Statistiken gilt grundsätzlich, dass diese mit und ohne Source beziehungsweise Destination gespeichert werden. Für die einzelnen Bereiche kann es Ausnahmen geben, welche dort explizit erwähnt werden. Es ergeben sich daraus die aus Tabelle 3 ersichtlichen, auch im weiteren Verlauf als solche bezeichnete, IP-Kombinationen.

Tabelle 3. Statistiken IP-Kombinationen

	Source nicht gesetzt	Source gesetzt
Destination nicht gesetzt	Alle Verbindungen	Alle Verbindungen von Source ausgehend
Destination gesetzt	Alle Verbindungen an Destination gehend	Alle Verbindungen zwischen dieser Source und Destination

Konkrete Werte

Bei den in der untenstehenden Tabelle enthaltenen Werten, handelt es sich um jene Werte, welche im Rahmen dieser Diplomarbeit aus den Echtdaten als Statistiken erstellt und anschließend gespeichert werden. Wie bereits oben erwähnt, wurde sich auf die Erstellung täglicher, monatlicher und jährlicher Statistiken geeinigt. Dies trifft auf sämtliche Werte zu.

Tabelle 4. Statistik-Werte

Verbindungsart	Wert	Speicherart
SNMP	Summe Datenmenge	Pro Klasse/Schule
SNMP	Durchschnitt Datenmenge	Pro Klasse/Schule
VPN	Summe Datenmenge	Pro IP-Kombination
VPN	Anzahl Verbindungen	Pro Protokoll
VPN	Anzahl Verbindungen	Pro Typ
Portscans	Anzahl Verbindungen	Pro IP-Kombination
DNS-Tunneling	Summe Datenmenge	Pro IP-Kombination
DNS-Tunneling	Anzahl Verbindungen	
DNS-Abfragen	Anzahl Verbindungen	Alle Verbindungen Von Source ausgehend
Login-Versuche	Anzahl Verbindungen	Pro Typ (erfolgreich/misslungen)
NetFlow	Anzahl Verbindungen	Pro IP-Kombination
NetFlow	Anzahl Verbindungen	Pro IP-Kombination Pro Protokoll
NetFlow	Anzahl Verbindungen	Pro IP-Kombination Pro Typ (Kategorie)
Programme	Summe Datenmenge	Pro IP-Kombination
Programme	Anzahl Verbindungen	Pro Programm
IP	Anzahl Verbindungen	An Destination gehend Von Source an Destination gehend
Rogue-DHCP-Server	Anzahl Verbindungen	Alle Verbindungen Von Source ausgehend

8.2.4 Umsetzung der Statistiken-Erstellung

Zur Erstellung der Statistiken wurde ein Python-Skript erstellt, welches verschiedene Methoden zum Erzeugen der Werte der einzelnen Bereiche bietet. Dieses wurde anschließend zur automatischen Ausführung eingerichtet.

8.2.4.1 Skript zur Erstellung

Das in Python programmierte Skript[12] verwendet die PyMongo-Library zum Auslesen der benötigten Daten aus der Echtdaten-Datenbank und die MySQL-Connector-Library zum Befüllen der Statistik-Datenbank. Hierfür werden die beiden Libraries mittels folgender Zeilen importiert:

```
import mysql.connector as connector
from pymongo import MongoClient
```

Listing 74. Python-Code: Importieren der Datenbank-Libraries

Anschließend können entsprechende Verbindungen zu den Datenbanken hergestellt werden. Der Umgang mittels PyMongo kann aus Kapitel 8.1.4 oben entnommen werden. Da es sich um zwei Star-Schemen handelt, gibt es auch zwei separate Datenbanken. Folgendermaßen wird eine Verbindung zu diesen hergestellt und gespeichert.

```
db_snmp = connector.connect(host=ip_address, user=user,
                            password=password, database="snmp")
db_con = connector.connect(host=ip_address, user=user,
                           password=password, database="connections")
```

Listing 75. Python-Code: Verbindung zu MySQL Server herstellen

Cursor-Objekte dienen zur Interaktion mit dem MySQL-Server. Sie ermöglichen das Ausführen von SQL-Anweisungen. Da für viele Operationen im Skript nicht sämtliche zurückgelieferte Ergebnisse benötigt werden, macht das Skript von *Buffered Cursors* Gebrauch. Ein solcher wird folgendermaßen angelegt:

```
cursor = db.cursor(buffered=True)
```

Listing 76. Python-Code MySQL-Cursor erstellen

Die auszuführende Operation wird in der typischen SQL-Notation in einer Variablen gespeichert. An diese zu übergebende Werte werden ebenfalls in einer Variablen – einem Tupel – abgespeichert und dem Cursor folgendermaßen übergeben:

```
cursor.execute(sql, val)
```

Listing 77. Python-Code: Ausführen von SQL-Operationen

Nach dem Einfügen oder Ändern von Datensätzen müssen diese bestätigt werden. Hierfür muss nach dem Ausführen der Änderungen ein Commit durchgeführt werden.

[12] Das vollständige Skript ist als externe Datei „GenerateStatistics.py“ auf der CD des Bibliotheksexemplars hinterlegt.

```
db.commit()
```

Listing 78. Python-Code: Ausführen der Änderungen

Argparse

Sämtliche Statistiken lassen sich über eigens erstellte Attribute erstellen. Dies verkompliziert zwar das Aufrufen all dieser gegebenen Attribute, stellt jedoch sicher, dass falls es bei einem Bereich zu einem Fehler kommt, die restlichen Statistiken nichtsdestotrotz erstellt und eingefügt werden. Beispielsweise wird durch das Attribut **`-snmp`** die Methode **`create_snmp_statistics()`** aufgerufen.

Befüllen der TIME-Dimension

Die TIME-Dimension muss vor dem Erstellen der tatsächlichen Statistiken laufend mit entsprechenden Datensätzen gefüllt werden. Je nach dem Zeitpunkt, an welchem die Echtdaten erfasst wurden, muss ein Datensatz erstellt werden. Der Methode **`create_time()`** müssen hierfür der Tag, das Monat und das Jahr übergeben werden.

Zuerst muss mittels eines Selects überprüft werden, ob entsprechende Datensätze – für den Tag, das Monat oder das Jahr – bereits vorhanden sind. Der Select und die dazugehörigen übergebenen Werte sehen wie folgt aus:

```
select = "SELECT * FROM time t WHERE t.year = %s AND t.month = %s AND " \
         "t.day = %s"
val = (year, month, day)
```

Listing 79. Python-Code: Select auf TIME-Dimension

Wird kein übereinstimmender Eintrag gefunden, muss ein neuer erstellt werden. Mittels der SQL-Funktion **`UUID()`** wird für die TIME-Dimension ein zufälliger eindeutiger *Primärschlüssel* erzeugt. Ähnlich dem Select ergeben sich dann die beiden Variablen:

```
if time_entry is None:
    sql = "INSERT INTO time (pk_time_id, year, month, day) " \
          "VALUES (UUID(), %s, %s, %s)"
    val = (str(date.year), str(date.month), str(date.day))
```

Listing 80. Python-Code: Einfügen von Datensätzen in Time-Dimension

Das sich daraus ergebende Statement muss anschließend ausgeführt werden. Sowohl das Überprüfen auf Vorhandensein sowie gegebenenfalls das Erstellen muss ebenfalls für das Monat und das Jahr durchgeführt werden. Ein Monatsdatensatz zeichnet sich durch den Wert **`0`** im Tagesfeld aus und ein Jahresdatensatz durch den Wert **`0`** im Tages- und Monatsfeld.

Gleichzeitig wird die Methode aus „Abfragen der TIME-Dimension“ zum Abfragen der Datensätze aus der TIME-Dimension aufgerufen. Das Ergebnis wird weiter zurückgeliefert.

Abfragen der TIME-Dimension

Die Funktion **`get_time()`** wird verwendet, um den *Primärschlüssel* der benötigten Datensätze abzufragen. Der Methode werden der Tag, das Monat und das Jahr übergeben. Anschließend werden die *Primärschlüssel* des Tages, des Monats und des Jahres mittels eines Selects abgefragt und zurückgeliefert. Der Select ähnelt dabei dem aus dem Teil „Befüllen der TIME-Dimension".

Allgemeines Erstellen der Statistiken

Vor dem tatsächlichen Erstellen der Statistiken, werden alle vorhandenen Datensätze, welche weder ausgewählt (**`selected`** gleich **`False`**) noch verarbeitet (**`processed`** gleich **`False`**) sind, für die Erstellung der Statistiken ausgewählt (**`selected`** auf **`True`** gesetzt).

```
query = {"processed": False, "selected": False}
new_values = {"$set": {"selected": True}}
col_snmp_entries.update_many(query, new_values)
```

Listing 81. Python-Code: Datensätze als ausgewählt markieren

Die Statistiken kommen durch das *Aggregieren* der Echtdaten aus der Echtdaten-Datenbank zustande. Hierfür wird in allen Bereichen auf unverarbeitete und ausgewählte Datensätze gefiltert. Der Aggregier-Methode von PyMongo wird dazu folgendes *Dictionary* übergeben:

```
{"$match": {"processed": False, "selected": True}}
```

Listing 82. Python-Code: Match für zu verarbeitende Daten

Weiters werden sämtliche Datensätze nach dem Jahr, dem Monat und dem Tag sowie weiteren für den Bereich benötigten Elementen gruppiert. Dies wird der Methode von PyMongo mit den dazugehörigen zu errechnenden Werten ebenfalls in einem *Dictionary* übergeben.

Nach dem Verarbeiten der einzelnen Datensätze, müssen diese auch als solches markiert werden. Hierfür wird das Feld **`processed`** der zuvor eingearbeiteten Dokumente auf den Wert **`True`** gesetzt.

```
query = {"processed": False, "selected": True}
new_values = {"$set": {"processed": True}}
col_snmp_entries.update_many(query, new_values)
```

Listing 83. Python-Code: Datensätze als verarbeitet markieren

Der Ablauf des Skripts lässt sich am besten durch Abbildung 32 beschreiben. Handelt es sich um NetFlow-, VPN- oder Programm-Statistiken werden variabel weitere Verbindungseinträge in der CONNECTION-Dimension ergänzt. Weiters müssen gegebenenfalls IP-Adressen aus den Datensätzen in der Datenbank eingefügt werden.

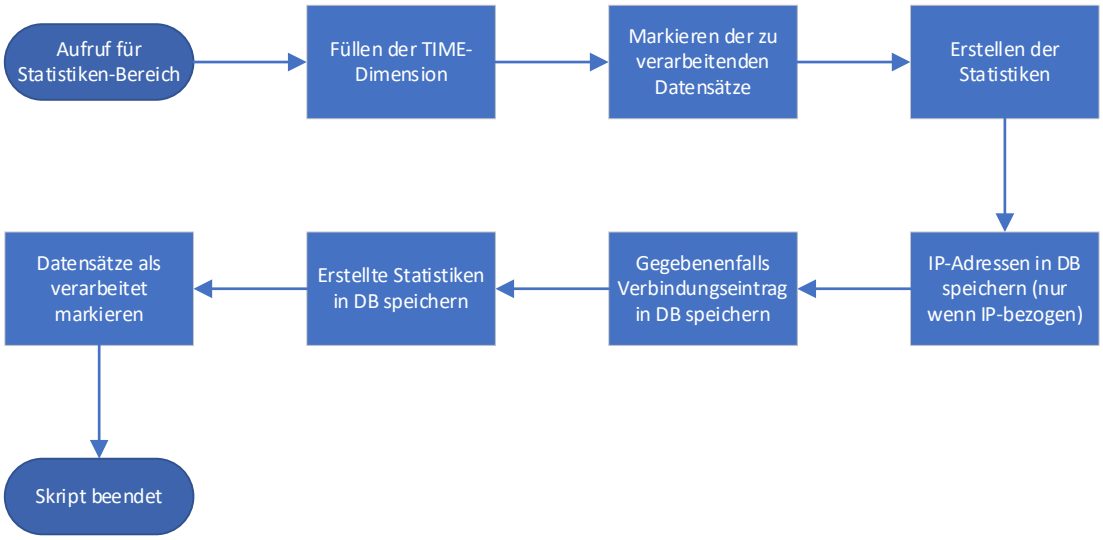


Abbildung 32. Ablauf: Statistiken-Erstellung

Einfügen der SNMP-Statistiken

Die Methode **`insert_snmp_statistics()`** speichert die übergebenen Statistiken in der Statistik-Datenbank ab. Zuerst werden durch das Aufrufen der in „Befüllen der TIME-Dimension“ beschriebenen Methode die benötigten *Primärschlüssel* gespeichert.

Der SQL-Insert in Listing 84 fügt bei nicht Vorhandensein der *Fremdschlüssel*-Kombination die gerade überlieferten Daten ein. Sind bereits Statistiken für den Tag, das Monat oder das Jahr vorhanden und für die Klasse ein Eintrag vorhanden, so werden diese entsprechend aktualisiert.

```
sql = "INSERT INTO facts (fk_time_id, fk_class_id, data_sum, " \
      "data_avg, total) VALUES (%s, %s, %s, %s, %s) " \
      "ON DUPLICATE KEY " \
      "UPDATE data_sum = data_sum + %s, total = total + %s, " \
      "data_avg = data_sum/total"
```

Listing 84. Python-Code: Einfügen von SNMP-Statistiken

Die für das Statement benötigten Werte werden in einem *Tupel* abgespeichert. Anschließend wird das Insert entsprechend ausgeführt.

Erstellen der SNMP-Statistiken

Vor dem Einfügen der SNMP-Statistiken werden diese mithilfe der Methode **`create_snmp_statistics()`** erstellt. Die Daten werden hierfür aus der MongoDB-*Collection* **`snmpentries`** herangezogen. Diese werden entsprechend der aus der Planung hervorgehenden Anforderungen aggregiert.

Neben dem Datum werden die SNMP-Datensätze auch nach dem Klassenraum gruppiert. Für diese werden dann jeweilig die Summe, der Durschnitt der Datenmenge und die Anzahl der Datensätze berechnet.

Diese Berechnungen werden mithilfe von *Feldoperationen* durchgeführt. Durch den Operator **`$sum`** wird in diesem Beispiel angegeben, dass die Summe von dem Feld **`traffic`**

aufsummiert werden muss. Wird dem gleichen Operator die Zahl **1** übermittelt, wird die Anzahl der gematchten Datensätze zurückgeliefert.

```
stats = list(col_snmp_entries.aggregate([
            {"$match": {"processed": False}},
            {"$group": {"_id": {"classroom": "$classroom_id",
                                "year": {"$year": "$timestamp"},
                                "month": {"$month": "$timestamp"},
                                "day": {"$dayOfMonth": "$timestamp"}},
                        "sum": {"$sum": "$traffic"},
                        "avg": {"$avg": "$traffic"},
                        "total": {"$sum": 1}}}
        ]))
```

Listing 85. Python-Code: SNMP-Statistiken erstellen pro Klassenraum

Dasselbe Statement wird mit **None** als **classroom** durchgeführt. Der Unterschied des Ergebnisses liegt darin, dass es sich dabei nicht um die Statistiken pro Klassenraum handelt, sondern um die der gesamten Schule – also die sämtlicher Klassenräume.

```
stats += list(col_snmp_entries.aggregate([
            {"$match": {"processed": False}},
            {"$group": {"_id": {"classroom": None,
                                …
            }}}
        ]))
```

Listing 86. Python-Code: SNMP-Statistiken erstellen

Die nun in einer Liste gespeicherten Statistiken werden anschließend durch die in „Einfügen der SNMP-Statistiken“ beschriebenen Methode in der Statistik-Datenbank abgespeichert.

Erstellen der Portscan-Statistiken

Die Portscan-Statistiken werden aus der *Collection* **portscanentries** der Echtdaten-Datenbank aggregiert. Die Methode **create_portscan_entries()** dient dem Erstellen der tatsächlichen Werte der Statistiken.

Entsprechend der Planung muss die Anzahl der durchgeführten Portscans im Schulnetzwerk pro IP-Kombination errechnet werden. Beispielsweise wird die Anzahl der Portscans von bestimmten Sources an beliebige Destinations anhand einer *Aggregation* ausgemacht. Bei dieser wird zwar nach der Source gruppiert, nicht aber nach der Destination. Bei der Gruppierung wird somit das Feld **dest** nach **None** gruppiert. Diese Kombinationen werden in einem *Tupel* gespeichert. In einer Schleife wird Kombination für Kombination durchgegangen. Anschließend wird für die Berechnung der Anzahl der Feldoperator **$sum** verwendet. Die Summe der Daten ist bei Portscans nicht relevant und wird somit **0** aufsummiert.

```
stats += list(col_portscan_entries.aggregate([
            {"$match": {"processed": False}},
            {"$group": {
                "_id": {"src": combination[0], "dest": combination[1],
                        "year": {"$year": "$start"},
                        "month": {"$month": "$start"},
                        "day": {"$dayOfMonth": "$start"}},
                "anz": {"$sum": 1}, "sum": {"$sum": 0}}}
        ]))
```

Listing 87. Python-Skript: Portscan-Statistiken erstellen

Einfügen der Portscan-Statistiken

Die Methode zum Einfügen der Portscan-Statistiken fällt etwas simpler aus. Mittels eines Selects wird der *Primärschlüssel* des Portscan-Verbindungseintrags aus der CONNECTION-Dimension abgefragt. Der Name der Verbindung lautet entsprechend **`Portscan`**. Der *Primärschlüssel* dieses Eintrags wird dann neben dem *Cursor* und den Statistiken an die Methode aus „Einfügen der Verbindungs-Statistiken“ übergeben.

Erstellen der NetFlow-Statistiken

NetFlow-Statistiken werden durch das Aggregieren der Datensätze aus der *Collection* **`netflowentries`** der Echtdaten-Datenbank gewonnen. Mittels der **`create_netflow_statistics()`** Methode werden also sämtliche Statistik-Werte erstellt.

Neben der Gruppierung nach IP-Kombinationen gruppiert die Methode die Anzahl der aufgebauten Verbindungen auch nach Kategorie und Protokoll. Hierfür sind die Kategorie und das Protokoll ebenfalls Teil der ID der Aggregation. Die Vorgehensweise ist dabei sehr ähnlich zu der aus „Erstellen der Portscan-Statistiken“.

Einfügen der NetFlow-Statistiken

Die zum Einfügen der Statistiken erstellte Methode **`insert_netflow_statistics()`** ermittelt zunächst den *Primärschlüssel* der CONNECTION-Dimension. Hierfür muss zuerst überprüft werden, ob der Typ oder das Protokoll in dem einzufügenden Datensatz gesetzt ist. Ist dies der Fall, wird mittels Selects überprüft, ob ein derartiger Connection-Datensatz vorhanden ist. Dann werden die Statistiken und der *Primärschlüssel* der Methode aus „Einfügen der Verbindungs-Statistiken“ übergeben. Ansonsten muss der entsprechende Connection-Eintrag erstellt und eingefügt werden.

Erstellen der restlichen Statistiken

Natürlich gibt es für sämtliche Bereiche entsprechende Methoden, welche die Statistiken berechnen. Diese orientieren sich an einem für den Bereich entsprechenden ähnlichen Konzept. Zum Schluss wird die jeweilige Einfüge-Methode verwendet, um die Werte in der Statistik-Datenbank speichern zu können.

Einfügen der restlichen Statistiken

Die Methoden finden den für die Connection entsprechenden *Primärschlüssel* heraus. Je nach Bereich müssen eventuell Datensätze variabel für Protokoll und Typ erstellt werden. Am Ende ruft die jeweilige Methode die Methode aus „Einfügen der Verbindungs-Statistiken“ auf.

Einfügen von IP-Adressen

Da die IP-Adressen in der Echtdaten-Datenbank ohnehin als Integer gespeichert werden, muss sich nicht um das Umrechnen gekümmert werden. Neben der IP-Adresse als Integer speichert die Methode `insert_ip()` den dazugehörigen Ländercode, welcher aus der Echtdaten-Datenbank abgefragt wird.

Einfügen der Verbindungs-Statistiken

Die Methode `insert_connection_statistics()` stellt die universale Möglichkeit, neben den SNMP-Statistiken die restlichen Werte in der Datenbank speichern zu können, zur Verfügung. Dieser Methode werden die statistischen Werte, der *Primärschlüssel* des entsprechenden Verbindungsdatensatzes sowie ein Cursor übergeben. Sie wird von den einzelnen Bereichsmethoden zum Einfügen verwendet.

Sofern ein Datensatz nicht nach der Source oder Destination gruppiert ist, muss dieser Wert auf `None` gesetzt werden. Daher überprüft die Methode vor dem Einfügen der Datensätze, ob der Wert gesetzt ist. Ist dies nicht der Fall, so wird diese IP-Adresse der Methode aus „Einfügen von IP-Adressen“ übergeben, um diese entsprechend zu speichern. Ist aber der Wert einer IP-Adresse auf `None` gesetzt, wird dieser stattdessen auf `0` gesetzt.

Anschließend werden durch das Aufrufen der in „Befüllen der TIME-Dimension“ beschriebenen Methode die benötigten *Primärschlüssel* der TIME-Datensätze gespeichert. Somit sind sämtliche *Primärschlüssel* zum Einfügen der Statistiken vorhanden.

Zum Schluss müssen die Daten noch in die Statistik-Datenbank geschrieben werden. Ist bereits ein Datensatz mit denselben *Fremdschlüsseln* vorhanden, so werden die Daten entsprechend aktualisiert. Ist die Kombination dieser noch nicht existent, wird einfach ein neuer Eintrag mit den übergebenen Werten erstellt.

8.2.4.2 Automatische Erstellung

Im Laufe eines Tages werden ständig neue Datensätze in die Echtdaten-Datenbank geschrieben. Am Abend werden nur noch wenige bis gar keine Daten gespeichert und daher kann hier das aufwendigere Erstellen der Statistiken stattfinden. Da sich das Skript zur Statistiken-Erstellung auf einem Windows-Host befindet, wird hierfür ein *ScheduledJob* herangezogen.

PowerShell-Skript

Für das Ausführen der einzelnen Methoden gibt es ein PowerShell-Skript. Dieses ruft nacheinander die entsprechenden Methoden zur Erstellung der Statistiken auf. Kommt es beim

Speichern eines Datensatzes zu einem Fehler, werden die restlichen Statistiken trotzdem erstellt und gespeichert. Ein kleiner Ausschnitt dieses Skripts sieht wie folgt aus[13]:

```
C:\Users\Specto\Desktop\GenerateStatistics.py --snmp-statistics
C:\Users\Specto\Desktop\GenerateStatistics.py --vpn-statistics
C:\Users\Specto\Desktop\GenerateStatistics.py --portscan-statistics
…
```

Listing 88. Ausschnitt aus PowerShell-Skript zur Statistiken-Erstellung

ScheduledJob

Das oben angeführte Skript muss nun mittels eines *ScheduledJobs* automatisch aufgerufen werden. Das Skript soll einmal täglich um 20:00 mit Administratorrechten ausgeführt werden. Hierzu wird ein Trigger erstellt, der den *ScheduledJob* mit dem PowerShell-Skript-Aufruf einmal täglich um 20:00 startet.

```
$trigger = New-JobTrigger -Daily -At "08:00 PM" -DaysInterval 1
```

Listing 89. PowerShell: Trigger für ScheduledJob

Weiters gibt der Skriptblock den auszuführenden Befehl an. In diesem Fall handelt es sich um das PowerShell-Skript, wobei es reicht den Namen anzugeben, um dieses ausführen zu lassen.

```
$scriptblock = [scriptblock]::Create("C:\Users\Specto\Desktop\
createStatistics.ps1")
```

Listing 90. PowerShell: Skriptblock für ScheduledJob

Außerdem muss festgelegt werden, dass das Skript mit Administratorrechten ausgeführt wird.

```
$options = New-ScheduledJobOption -RunElevated
```

Listing 91. PowerShell: Optionen für ScheduledJob

Die oben in Variablen gespeicherten Argumente werden beim Erstellen des *ScheduledJobs* angegeben. Mithilfe folgender Zeile kann der ScheduledJob **`Create-Statistics`** erstellt werden.

```
Register-ScheduledJob -Name "Create-Statistics" -ScriptBlock $scriptblock
-Trigger $trigger -ScheduledJobOption $options -Credential (Get-
Credential Specto)
```

Listing 92. PowerShell: Erstellen des ScheduledJobs

[13] Das vollständige Skript ist als externe Datei „createStatistics.ps1“ auf der CD des Bibliotheksexemplars hinterlegt.

8.2.5 Beispieldatensätze

Die geplanten Statistiken können nun in der Statistik-Datenbank gespeichert werden. Einzelne Datensätze können mittels *SQL* über beispielsweise DataGrip[14] testweise abgefragt werden.

Eine für die Medientechnik benötigte Abfrage könnte sein: „Wie viele Daten wurden von dem Klassenraum **277** im **Jahr 2020** gesamt und durchschnittlich produziert?“ Dabei handelt es sich durch den Star-Schema-Aufbau um einen sehr einfachen Select. Geht man von der *Faktentabelle* aus, sind zwei Join-Operationen notwendig – eine in die CLASS-Dimension und eine in die TIME-Dimension. Der Select sieht wie folgt aus:

```
SELECT data_sum, data_avg
FROM facts
         INNER JOIN time t ON facts.fk_time_id = t.pk_time_id
         INNER JOIN class c ON facts.fk_class_id = c.pk_class_id
WHERE t.day = 0
  AND t.month = 0
  AND t.year = 2021
  AND c.nr = 277
```

Listing 93. DataGrip: Select SNMP-Statistiken

Dieser liefert als Ergebnis die Werte der insgesamt und durchschnittlich übertragenen Daten zurück. Einen ähnlichen automatisch generierten Select führt dann die API für die Medientechnik durch. Das Ergebnis kann wie folgt aussehen:

```
2528587728.58,81569.71
```

Listing 94. DataGrip: Ausgabe SNMP-Statistiken

Eine weitere Abfrage aus einem anderen Bereich könnte die Anzahl der durchgeführten **Portscans** von der IP-Adresse **10.76.1.23** auf die IP-Adresse **10.76.1.254** im **Jänner 2021** sein. Für diesen Select sind ausgehend von der *Faktentabelle* vier Join-Operationen vonnöten. Der Select sieht wie folgt aus:

```
SELECT amount
FROM facts
         INNER JOIN time t ON facts.fk_time_id = t.pk_time_id
         INNER JOIN ip s ON facts.fk_source_ip = s.pk_ip
         INNER JOIN ip d ON facts.fk_destination_ip = d.pk_ip
         INNER JOIN connection c
             ON facts.fk_connection_id = c.pk_connection_id
WHERE t.day = 0
  AND t.month = 1
```

[14] IDE (Integrated Development Environment) für Datenbanken

```
    AND t.year = 2021
    AND c.name = 'Portscan'
    AND INET_NTOA(fk_source_ip) = '10.76.1.23'
    AND INET_NTOA(fk_destination_ip) = '10.76.1.254'
```

Listing 95. DataGrip: Select Portscan-Statistiken

Daraufhin bekommt man die Anzahl der oben beschriebenen durchgeführten Portscans zurückgeliefert. Ein ähnlicher automatisierter Select wird durch die API für die Medientechnik durchgeführt. In diesem Fall wird ein einzelner Wert, zum Beispiel `12`, zurückgeliefert.

9 File Sharing zwischen VMs

Die *Packet Inspection* des Schulnetzverkehrs wird in vielen Bereichen der Diplomarbeit „Specto“ benötigt. Damit nicht mehrere Maschinen direkt in das Schulnetzwerk angebunden werden müssen, erstellt eine zentrale Maschine *PCAP*-Dateien. Diese Dateien müssen automatisiert an alle erforderlichen Maschinen verteilt werden.

9.1 Verteilen der PCAP-Dateien

Die Ausgangssituation sieht folgendermaßen aus: Eine virtuelle Maschine wird direkt an die Netzwerkkomponenten der Schule angeschlossen und liest ausgewählten Netzverkehr mit. Diese Maschine wird von dem Partnerteam Cerebrum bereitgestellt und wird in der Folge *Sniffer-VM* genannt. Aus den mitgelesenen Daten werden automatisch *PCAP*-Dateien erstellt, welche anschließend in anderen Teilbereichen ausgewertet werden können.

Die auszuwertenden Dateien müssen 48 Stunden verzögert und an alle Client-VMs des Teams „Specto“ gelangen, damit sie entsprechend verarbeitet werden können. Dafür ergibt sich die vereinfachte Topologie in Abbildung 33.

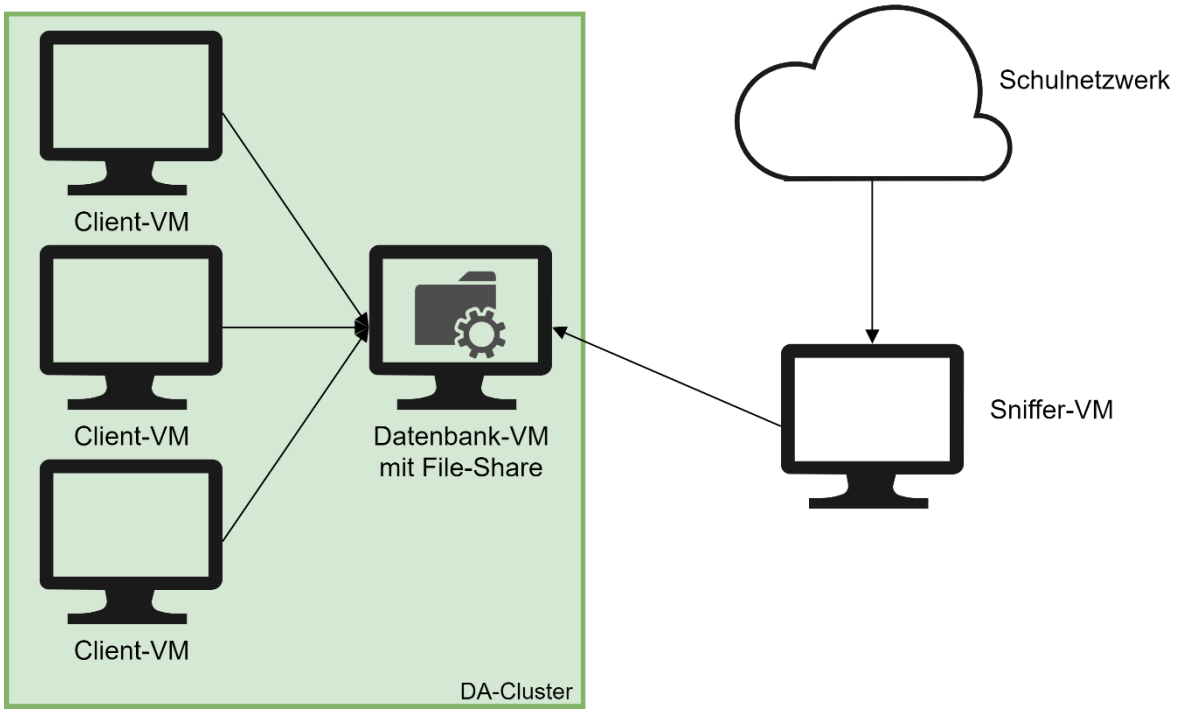


Abbildung 33. Topologie für die Verteilung der PCAP-Dateien

9.1.1 Ablauf

Der Topologie in Abbildung 33 kann entnommen werden, dass die Daten des Schulnetzwerks zunächst mittels der *Sniffer-VM* mitgelesen und gespeichert werden. Diese gespeicherten *PCAP*-Dateien werden anschließend in den File-Share der *Share-VM* hochgeladen. Von dort können alle Mitglieder auf die *PCAP*-Dateien zugreifen und diese lokal kopieren.

Natürlich muss dieser Ablauf zeitlich geregelt werden, denn die PCAP-Dateien müssen aus Speichergründen periodisch vom File-Share gelöscht werden. Zum Verständnis dient das Ablaufdiagramm in Abbildung 34.

Auf der Client-Seite wird alle fünf Minuten abgefragt, ob sich *PCAP*-Dateien im Ordner *PCAP* befinden. Falls ja werden diese Dateien in einen lokalen Ordner der Client-Maschine kopiert. Falls nein geschieht nichts. In beiden Fällen fragt die Client-Maschine nach fünf Minuten erneut den Ordner am File-Share ab.

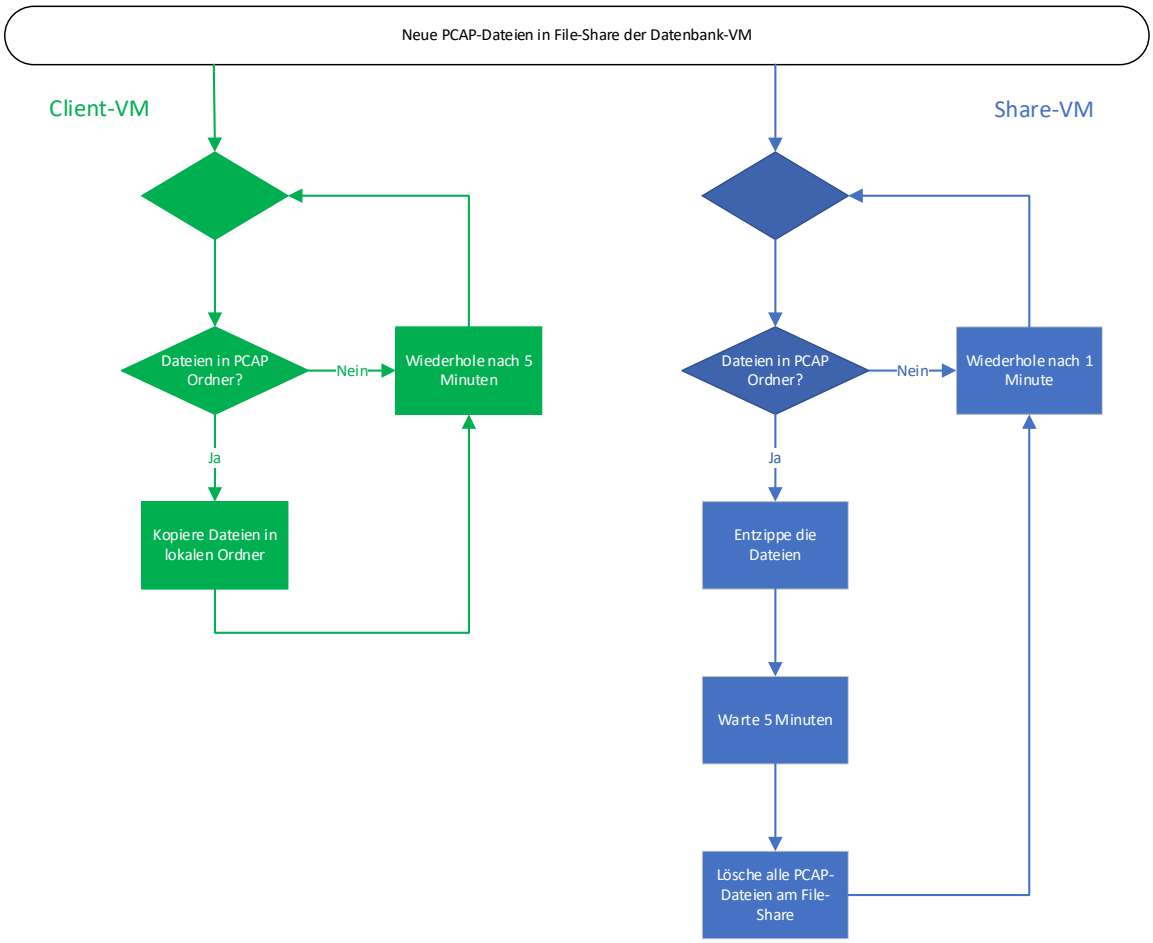


Abbildung 34. Ablauf für die Verteilung der PCAP-Dateien

Die *Share-VM* hingegen fragt minütlich ab, ob sich Dateien im Ordner *PCAP* befinden. Falls ja werden diese entzippt und anschließend wird fünf Minuten gewartet. In diesem Zeitraum sollten die Client-VMs *PCAP*-Dateien auf dem File-Share entdecken und diese lokal speichern. Nach Ablauf dieser 5 Minuten löscht die *Share-VM* den Inhalt des *PCAP*-Ordners. Falls sich keine Dateien im Ordner *PCAP* befinden, wird nach einer Minute erneut der Ordnerinhalt abgefragt.

9.1.2 Umsetzung

Um den in Abbildung 34 gezeigten Ablauf zu automatisieren, werden auf den Maschinen Shell-Skripte geschrieben. Damit diese Shell-Skripte automatisch aufgerufen werden, werden unter Linux *Cronjobs* und unter Windows *ScheduledJobs* verwendet.

9.1.2.1 Sniffer-VM

Die *Sniffer-VM* erfasst Netzwerkaktivitäten direkt aus dem Schulnetzwerk mittels eines *Mirror-Ports*. Die Maschine erstellt daraus *PCAP*-Dateien und leitet diese zur *Share-VM* weiter.

Damit die *Sniffer-VM* die erstellten Daten weiterleiten darf, müssen die Daten aus Datenschutzgründen 48 Stunden auf der Maschine verwahrt werden. Daher wird ein Shell-Skript erstellt, dass alle *PCAP*-Dateien die älter als zwei Tage sind an die Share-VM weiterleitet. Dieses Skript iteriert durch alle Dateien im Ordner **`/specto/files`** und sieht nach, ob diese Dateien älter als 48 Stunden sind. Sind die Dateien älter, werden sie auf einen Ordner der *Share-VM* hochgeladen und anschließend lokal gelöscht.

Der Inhalt der Iteration sieht im Shell-Skript[15] wie folgt aus:

```
if test $(find "/specto/files/$file" -mtime +2)
   then
         sudo smbclient //10.70.8.1/share --user specto gansgeheim123! -c
‘cd PCAP;put /specto/files/’$file’ ‘$file
         sudo rm /specto/files/$file
   fi
```

Listing 96. Linux Shell 48-Stunden-alte PCAP-Dateien verarbeiten

Damit dieses Skript automatisch aufgerufen wird, muss ein neuer Eintrag in der crontab erstellt werden. Dazu wird die Crontabelle wie in Listing 40 geöffnet und unten wird folgende Zeile angehängt.

```
@hourly bash /specto/.pushData.sh
```

Listing 97. Linux Shell-Skript jede Stunde aufrufen

Das zuvor erklärte Skript wird durch den Cronjob stündlich aufgerufen und überprüft automatisch, ob es *PCAP*-Dateien gibt, die älter als 48 Stunden sind.

Dieselbe Konfiguration wurde für den *SNMP-Host* und den *Syslog Server* angewendet, wobei hier das Skript alle 5 Minuten aufgerufen wurde.

9.1.2.2 Share-VM

PCAP-Dateien dürfen nicht mehrmals vom File-Share der Share-VM von den Clients lokal gespeichert werden. Daher müssen alle PCAP-Dateien wieder nach einem bestimmten Zeitraum vom File-Share gelöscht werden.
Hierzu wird der Ablauf aus Abbildung 34 in Form eines PowerShell-Skripts umgesetzt. Dieses Skript überprüft, ob der Ordner *PCAP* Dateien beinhaltet.

[15] Das vollständige Skript ist als externe Datei „pushData.sh“ auf der CD des Bibliotheksexemplars hinterlegt.

```
if((Get-ChildItem C:\share\PCAP\ | Measure-Object).Count -gt 0){
...
}
```

Listing 98. PowerShell Anzahl der Dateien in einem Ordner überprüfen

Befinden sich Dateien im Ordner, wird überprüft, ob es sich um gezippte Dateien handelt. Diese Dateien sind gezippt, da sie aus Speichergründen sonst zu viel Volumen auf der *Sniffer-VM* verbrauchen würden. Gezippte Dateien werden in ihre Ursprungsdateien entpackt und die gezippte Datei wird verworfen.

```
Get-ChildItem -Path C:\share\PCAP\ | Where-Object {($_.Extension -like
"*.zip")}|ForEach-Object{
        Expand-Archive -Path ($_.DirectoryName+"\"+$_.BaseName+".zip") -
DestinationPath C:\share\PCAP\ -Force
        $_.Delete()
    }
```

Listing 99. PowerShell gezippt Dateien entpacken und Quelle löschen

Um den Client-VMs laut Abbildung 34 genug Zeit zu geben, wird noch 5 Minuten gewartet, bis alle entpackten *PCAP*-Dateien gelöscht werden.

Damit dieses PowerShell-Skript automatisiert werden kann, muss unter Windows anstatt eines *Cronjobs* ein *Scheduled-Job* erstellt werden. Ein Scheduled-Job besteht aus einigen Variablen, die zuerst erstellt werden müssen:

- Ein Trigger, der angibt, wie oft und wann ein Skript ausgeführt werden soll:

```
$trigger = New-JobTrigger -Once -At (Get-Date) -RepetitionDuration
([System.TimeSpan]::MaxValue) -RepetitionInterval (New-TimeSpan -Minutes
1)
```

Listing 100. PowerShell Trigger für Scheduled-Job erstellen

- Ein Skript-Block, der angibt welche Befehle bzw. Skripte ausgeführt werden sollen:

```
$scriptblock = [scriptblock]::Create(“removePCAP.ps1“)
```

Listing 101. PowerShell Skript-Block für Scheduled-Job erstellen

- Optionen, welche ein bestimmtes Verhalten des Schedule-Jobs einstellen:

```
$options = New-ScheduleJobOption -RunElevated -ContinueIfGoingOnBattery –
StartIfOnBattery
```

Listing 102. PowerShell Optionen für Scheduled-Job einstellen

Durch Erstellen dieser drei Variablen kann der Scheduled-Job erstellt werden:

```
Register-ScheduleJob -Name “Remove PCAPs“ -ScriptBlock $scriptblock -
Trigger $trigger -ScheduleJobOption $options -Credential (Get-Credential
Specto)
```

Listing 103. PowerShell Scheduled-Job mithilfe von Variablen erstellen

Der erstellte Scheduled-Job wird minütlich aufgerufen und erfüllt die Aufgaben aus Abbildung 34.

Falls nötig, kann der *Scheduled-Job* im Windows-integrierten Programm *Task-Scheduler* bzw. *Aufgabenplanung* angesehen werden.

9.1.2.3 Client-VMs

Während die Share-VM erhaltene *PCAP*-Dateien entzippt und wartet, haben die Client-Maschinen Zeit diese lokal zu speichern. Da die Client-VMs sowohl Windows als auch Linux als Betriebssystem verwenden, muss das automatische Speichern der PCAP-Dateien vom File-Share für beide Betriebssysteme durchgeführt werden.

Unter Linux ist die Implementation dieses Ablauf u, einiges simpler als unter Windows. In Linux muss lediglich folgender Befehl ausgeführt werden, damit Dateien von der Share-VM lokal kopiert werden:

```
smbclient //10.70.8.1/share gansgeheim123! -c “prompt OFF;recurse ON;cd
PCAP;lcd /home/specto/Schreibtisch/PCAPs/;mget *“
```

Listing 104. Linux Shell Dateien von File-Share in lokalen Ordner speichern

Mit diesem Befehl wird vom SMB-Server 10.70.8.1 der Ordner *share* geöffnet. Mit dem Attribut -c können Befehle ausgeführt werden, so wird beispielsweise das Prompt abgeschaltet, Rekursion aktiviert, in den Ordner PCAP am Share gewechselt, der lokale Ordner PCAPs mit absolutem Pfad angegeben und alle Dateien in diesen lokalen Ordner heruntergeladen.

Um diesen Befehl zu automatisieren, muss nur ein Cronjob angelegt werden. Hierzu wird im Crontab ein neuer Eintrag hinzugefügt, der wie folgt aussieht:

```
*/5 * * * * smbclient //10.70.8.1/share gansgeheim123! -c “prompt
OFF;recurse ON;cd PCAP;lcd /home/specto/Schreibtisch/PCAPs/;mget *“
```

Listing 105. Linux Shell Befehl alle 5 Minuten ausführen

Damit werden, wie in Abbildung 34 festgelegt, alle 5 Minuten alle Dateien auf dem File-Share im Ordner *PCAP* lokal in **`/home/specto/Schreibtisch/PCAPs`** kopiert.

Unter Windows muss wie in 9.1.2.2 Share-VM ein *Scheduled-Job* erstellt werden, welcher alle 5 Minuten den folgenden Befehl ausführt.

```
Copy-Item -Path //10.70.8.1/share/PCAP/* -Destination
C:\Users\Specto\Desktop\PCAPs -Recurse -Force
```

Listing 106. PowerShell Dateien von File-Share in lokalen Ordner speichern

Dieser Befehl kopiert alle Dateien aus dem Ordner am File-Share und speichert sie lokal in einem beliebigen Ordner ab.

Damit dieser Befehl automatisch ausgeführt wird, muss unter Windows ein Schedule-Job erstellt werden. Dieser benötigt dieselben Variablen, die in 9.1.2.2 Share-VM aufgezählt werden.

Es müssen nur der Trigger aus Listing 100 auf 5 Minuten und der Skript-Block aus Listing 101 auf den Befehl aus Listing 106 abgeändert werden. Damit kann, genau wie in Listing 103 ein Scheduled-Job mit dem Namen *Collect PCAPs* erstellt werden.

Genau wie unter Linux können dadurch alle 5 Minuten PCAP-Dateien von einem File-Share lokal gespeichert werden.

9.2 Beschreiben der Echtdaten-Datenbank

In der Diplomarbeit „Specto“ erzeugt bzw. verarbeitet eine Vielzahl von virtuellen Maschinen Daten. Die verschiedenen Daten aus Tabelle 5 sollen anschließend in eine Echtdaten-Datenbank geschrieben werden.

Dabei ergibt sich das Problem, dass zwei unterschiedliche Maschinen nicht gleichzeitig die erfassten Daten direkt in die Datenstruktur schreiben können. Also muss der Zugriff auf die Echtdaten-Datenbank zeitlich geregelt werden. (vgl. Administrator 2009)

Tabelle 5. Teilbereiche von Specto

Name	Beschreibung
DNS External	Schulexternen DNS-Server werden erkannt
DNS-Tunneling	DNS-Tunneling im Netzwerkverkehr wird erkannt
IP	Zu öffentlichen IP-Adressen werden Informationen, wie AS-Nummer, erfasst
Login	Login-Versuche auf bestimmte Geräte wird erfasst
NetFlow	Traffic-Informationen, wie IP-Adresse, Port-Nummer, werden erfasst
Portscans	Portscans auf bestimmte Geräte werden erfasst
Programme	Die Benutzung einiger Programme wird erkannt
Rogue DHCP	Rogue DHCP-Server werden im Schulnetz erkannt
SNMP	Die verbrauchte Datenmenge wird mittels SNMP erkannt
VPN	Bestimmte VPNs werden erkannt

9.2.1 Ablauf

Der erste Ansatz wäre, nur jede virtuelle Maschine in einem bestimmten Zeitraum Daten in die Echtdaten-Datenbank schreiben zu lassen. Das heißt jede *VM* erhält einen Time-Slot in der die erfassten Daten in die Echtdaten-Datenbank gespeichert werden können.

Ist dieser Time-Slot vorbei, müssen die Daten wieder so lange zwischengespeichert werden, bis der Time-Slot wieder für diese VM „offen“ ist. Dies könnte mit Hilfe von Cronjobs umgesetzt werden. Um die Daten möglichst aktuell zu halten, empfiehlt es sich einen kleinen Zeitraum pro Time-Slot zu setzen.

Ein einfacherer Weg das gleichzeitige Beschreiben der Echtdaten-Datenbank zu verwalten, ist die erfassten Daten zunächst zentral in einer Ordnerstruktur zu speichern. Diese Ordnerstruktur liegt auf der Share-VM aus Abbildung 33. Wenn Daten in diese Ordnerstruktur verschoben werden, werden diese Daten in einer bestimmten Reihenfolge abgearbeitet.

Sind in einem Teilbereich gerade keine Dateien vorhanden, so wird dieser Bereich übersprungen. Das Überspringen würde die Ineffizienz aus dem ersten Ansatz verhindern, da dieser Zeitraum im ersten Ansatz nicht genutzt werden würde. Dieser Ansatz sieht für den Teilbereich *DNS External* wie in Abbildung 35 aus.

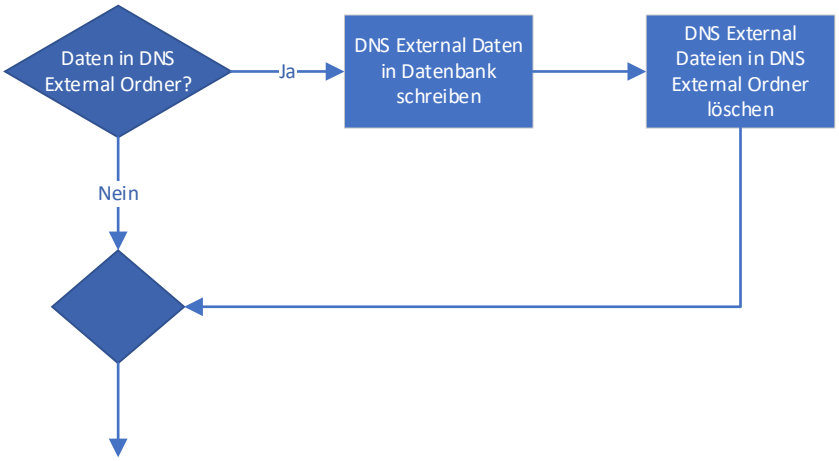


Abbildung 35. Ablauf für DNS External

Für jeden Datensatz, in Abbildung 35 konkret externe *DNS-Server*, wird zunächst nachgesehen, ob Dateien in der zentralen Ordnerstruktur vorhanden sind. Falls ja, werden die Daten ausgelesen und in die Echtdaten-Datenbank geschrieben. Anschließend werden die Dateien gelöscht.

Falls nein, wird dieser Bereich übersprungen und der nächste Datensatz wird verarbeitet. Dieser Ablauf wird minütlich aufgerufen und verarbeitet alle Daten der Teilbereiche aus Tabelle 5.

9.2.2 Umsetzung

Damit der Ablauf aus Abbildung 35 für jeden Teilbereich automatisiert werden kann, wird ein PowerShell-Skript[16] erstellt. Dieses Skript sieht beispielsweise für den Bereich *External DNS* wie folgt aus:

```
gci “C:\share\DNS External“|ForEach-Object{
    python3 C:\Users\Specto\Desktop\MongoDB.py save -dnst
C:\share\Tunneling\$_
}
```

Listing 107. PowerShell Ordnerinhalt verarbeiten und löschen

[16] Das vollständige Skript ist als externe Datei „insertData.ps1“ auf der CD des Bibliotheksexemplars hinterlegt.

Die erste Zeile listet alle Dateien im angegebenen Ordner auf und führt für jede gefundene Datei den Inhalt der Schleife aus. Innerhalb der Schleife wird ein Python-Skript des Teammitgliedes Patrick Krenn aus *8.1.4 Umsetzung des Skripts* aufgerufen. Dieses Python-Skript schreibt den Inhalt der Datei in die Echtdaten-Datenbank und löscht anschließend die Datei.

Der Code aus Listing 107 wird nun für alle Teilbereiche aus Tabelle 5 im PowerShell-Skript erweitert.

Damit dieses PowerShell-Skript automatisch jede Minute aufgerufen wird, wird ein *Scheduled-Job* erstellt. Dazu wird wie bei Listing 100 ein minütlicher Trigger erstellt. Beim Erstellen des Skript-Block aus Listing 101 wird das erwähnte PowerShell-Skript angegeben. Mit der zusätzlichen Angabe von den Optionen aus Listing 102 kann, genau wie in Listing 103, ein Scheduled-Job mit dem Namen *Insert-Data* erstellt werden.

Der Scheduled-Job *Insert-Data* liest nun, falls vorhanden, alle Dateien minütlich aus der Ordnerstruktur ein und schreibt diese in die Echtdaten-Datenbank.

10 API

Es wurde eine eigene *REST API* erstellt, die die Schnittstelle zwischen dem Partnerteam „Oculus“ und den durch die Teams „Nexus“ und „Cerebrum“ gewonnenen Daten darstellt. Die API ermöglicht es, über einheitliche Abfragen auf die Daten zuzugreifen. Somit müssen von Team „Oculus“ selbst keine komplizierten und unterschiedlichen Selects für die beiden DBMS[17] geschrieben werden. Damit dies umgesetzt werden konnte, wurden im ersten Schritt die einzelnen Teilbereiche, in denen Daten gesammelt werden, mit „Oculus“ durchbesprochen. Weiters anhand der zur Visualisierung benötigten Daten wurden der Aufbau und die Ausgaben der API definiert und festgehalten. Dies wurde sowohl für die Datensätze der Echtdaten-Datenbank sowie für die in der Statistik-Datenbank durchgeführt. Genaueres zum definierten Aufbau der API kann dem Kapitel „Planung der API“ entnommen werden. Für die Erstellung der API wurde Python mit der Library „Flask“ verwendet. In der nachfolgenden Grafik ist die Datenabfrage visuell dargestellt.

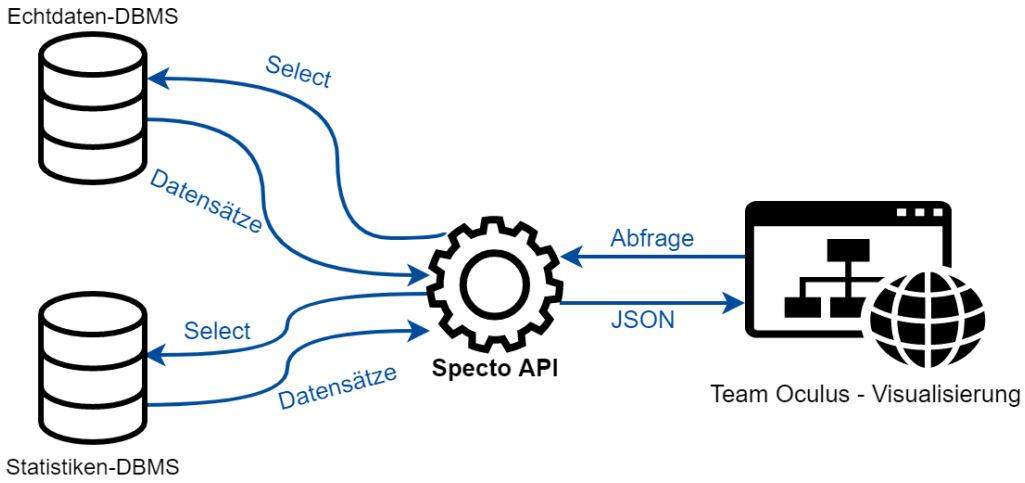


Abbildung 36. Specto API

Eine API (Application Programming Interface) stellt eine Schnittstelle für andere Programme zur Verfügung. Über definierte Anweisungen können bestimmte Daten ausgetauscht werden. Dabei spielt besonders die vordefinierte *Syntax* eine wichtige Rolle, da nur durch die Eingabe der richtigen Befehle auch die gewünschten Daten gewonnen werden können. Aufgrund dessen gibt es für APIs in den meisten Fällen entsprechende Dokumentationen, wie es auch im Fall der Diplomarbeit mit einem API-Konzept umgesetzt worden ist[18]. (vgl. Luber o. J.; vgl. IONOS 2020b)

Wie zu Beginn dieses Kapitels erwähnt, wurde für „Specto“ eine *REST* (Representational State Transfer) *API* erstellt. Diese lässt sich einfach über *URLs*, anstatt über *XML*-Anfragen,

[17] Datenbankmanagementsysteme
[18] Das vollständige API-Konzept ist als externe Datei „Specto_API_Konzept.pdf“ auf der CD des Bibliotheksexemplars hinterlegt.

abfragen. Dabei genügen in den meisten Fällen *HTTP*-Operationen, wie beispielsweise *GET*, *POST*, *PUT* und *DELETE*. Ebenso kann das Ausgabeformat selbst bestimmt werden und die Daten können wie im Fall der Diplomarbeit im *JSON-Format* zurückgeliefert werden. (vgl. IONOS 2020c; vgl. Drilling 2017)

10.1 Planung der API

In Meetings wurde in enger Zusammenarbeit mit Team „Oculus“ festgelegt, welche Daten von den einzelnen Themenbereichen für die Visualisierung benötigt werden und daher von der API bereitgestellt werden müssen. Die Daten der Echtdaten-Datenbank werden für interaktive Darstellungen verwendet, während die Daten der Statistik-Datenbank für das Anzeigen statistischer Grafiken benötigt werden. Somit haben sich für den **`time`**-Parameter folgende allgemeine Abfrage-Möglichkeiten ergeben:

- **aktuell** – liefert die zuletzt gespeicherten Datensätze beziehungsweise den zuletzt gespeicherten Datensatz im für den Bereich definierten Zeitraum zurück
- **custom** – liefert alle Datensätze innerhalb eines angegebenen Zeitraums zurück
- **history** – liefert die statistischen Datensätze (immer jährlich, monatlich und täglich) vom aktuellen Datum ausgehend oder unter Angabe eines bestimmten Datums

Unter Angabe von **`aktuell`** oder **`custom`** werden somit die Daten der Echtdaten-Datenbank abgefragt und über die Angabe von **`history`** die Daten der Statistik-Datenbank. Je nach Themenbereich können verschiedene weitere Parameter angegeben werden oder auch entfallen. Ebenso unterscheiden sich die Rückgaben. Wie die Definitionen für die einzelnen Bereiche festgelegt wurden, kann den folgenden Unterkapiteln entnommen werden. Grundsätzlich ist jede Abfrage nach dem folgenden Schema aufgebaut:

```
https://<IP-Adresse|Domäne>:5000/specto/<Bereich>?Parameter1&Parameter2…
```

Listing 108. Allgemeiner Abfrage-Aufbau

10.1.1 Klassen

Unter Angabe des Bereichs **`/classes`** und ohne Angabe weiterer Parameter können alle in der Echtdaten-Datenbank gespeicherten Klassen abgefragt werden. Es werden die Raumnummer als Integer-Wert und die Raumbezeichnung als String zurückgeliefert. Die Abfrage und eine Ausgabe hierzu können folgendermaßen aussehen:

```
https://<IP-Adresse|Domäne>:5000/specto/classes
```

Listing 109. Abfrage der Klassen

```
[
   {
         "nr": 277,
         "description": "5AX"
   },
   {
         "nr": 254,
         "description": "5CN"
   }
]
```

Listing 110. JSON: Beispielausgabe der Klassen

10.1.2 SNMP

Für den SNMP-Bereich werden zum einen die Datenmengen der einzelnen Klassen und zum anderen die Summe und der Durchschnitt der Datenmengen abgefragt. Der Bereich hat die Bezeichnung **`/snmp`**. Damit die einzelnen Klassen abgefragt werden können, gibt es den zusätzlichen Parameter **`room=<Integer>`**, dem die entsprechende Raumnummer übergeben wird.

Echtdaten

Wird dem time-Parameter der Parameter **`aktuell`** übergeben, wird der zuletzt in der Echtdaten-Datenbank gespeicherte Datenmengen-Datensatz (Dezimal) in Bytes zurückgeliefert. Die Abfrage hierzu kann wie folgt aussehen:

```
https://<IP-Adresse|Domäne>:5000/specto/snmp?room=277&time=aktuell
```

Listing 111. SNMP time=aktuell Abfrage

Wenn **`custom`** für **`time`** angegeben wird, muss zusätzlich der Parameter **`moment=%d.%m.%Y %H:%M:%S`** übergeben werden. Von dem hier übergebenen Zeitpunkt ausgehend wird der zuletzt gespeicherte Datenmengen-Datensatz (Dezimal) in Bytes zurückgeliefert. Eine Abfrage sieht beispielsweise so aus:

```
https://<IP-
Adresse|Domäne>:5000/specto/snmp?room=277&time=custom&moment=23.11.2020
20:06:00
```

Listing 112. SNMP time=custom Abfrage

Das Ausgabeformat unterscheidet sich nicht durch die angeführten Werte, die dem time-Parameter übergeben werden und sieht beispielshaft wie folgt aus:

```
{
   "traffic": 9253.129
}
```

Listing 113. JSON: SNMP Echtdaten-Ausgabe

Statistiken

Wird als Zeitwert **history** angegeben, muss zusätzlich der Parameter **type=** angegeben werden. Bei der Übergabe von **aktuell** oder **custom** (benötigt zusätzlich den Parameter **moment=%d.%m.%Y**) werden die statistischen Datensätze ausgehend vom aktuellen Datum beziehungsweise dem übergebenen Datum zurückgeliefert. Weiters besteht über die Angabe von **years|months|days** die Möglichkeit, alle vorhandenen Jahres-| Monats-| Tagesstatistiken abzufragen. Lässt man den zusätzlichen **room**-Parameter bei einer Abfrage weg, werden die Summe und der Durchschnitt der Datenmengen aller Räume, also der gesamten Schule, abgefragt.

Die Abfragen können somit wie folgt aussehen:

```
https://<IP-
Adresse|Domäne>:5000/specto/snmp?room=277&time=history&type=aktuell
```

Listing 114. SNMP Statistiken type=aktuell

```
https://<IP-
Adresse|Domäne>:5000/specto/snmp?room=277&time=history&type=custom&moment
=01.09.2020
```

Listing 115. SNMP Statistiken type=custom

```
https://<IP-
Adresse|Domäne>:5000/specto/snmp?room=277&time=history&type=years|months|
days
```

Listing 116. SNMP Statistiken type=years|months|days

Die Rückgaben für **aktuell** und **custom** entsprechen demselben Format und sehen wie folgt aus:

```
{
   "yearly": {
         "sum": 74720.6,
         "avg": 10674.4
   },
   "monthly": {…},
   "daily": {…}
}
```

Listing 117. JSON: SNMP Statistiken-Ausgabe type=aktuell | custom

Die Ausgabe unterscheidet sich bei der Angabe von **years|months|days** dahingehend, dass zusätzlich, je nach gewählter Option, das Jahr, das Monat und der Tag auch zurückgeliefert werden. Somit sieht beispielsweise die Ausgabe, wenn **type=days** angegeben wird, so aus:

```
[
    {
        "year": 2021,
        "month": 1,
        "day": 9,
        "sum": 32023.1,
        "avg": 10674.4
    },
    {
        "year": 2021,
        "month": 1,
        "day": 8,
        "sum": 10674.4,
        "avg": 10674.4
    }
]
```


Listing 118. JSON: SNMP Statistiken-Ausgabe type=days

10.1.3 NetFlow

Bei NetFlow stehen nur die Optionen **`custom`** und **`history`** für den type-Parameter zur Verfügung. Die Abfrage erfolgt über den Bereichsnamen **`/netflow`**.

Echtdaten

Für die Echtdaten-Abfrage steht nur **`custom`** zur Verfügung. Hier werden zusätzlich die Parameter **`start=%d.%m.%Y %H:%M:%S`** und **`end=%d.%m.%Y %H:%M:%S`** benötigt. Als Ergebnis wird die Gesamtanzahl der Verbindungen (Integer) im angegebenen Zeitraum zurückgeliefert. Zusätzlich gibt es einen **`protocol`**-Parameter, über welchen **`TCP`**, **`UDP`** oder **`other`** angegeben werden kann, um so die Verbindungsanzahl gefiltert nach einem der Protokolle zu bekommen. Damit stehen für die Echtdaten folgende zwei beispielhafte Abfragen zur Verfügung:

```
https://<IP-
Adresse|Domäne>:5000/specto/netflow?time=custom&start=01.10.2020
17:19:20&end=13.12.2020 18:19:20
```

Listing 119. NetFlow Echtdaten-Abfrage ohne Protokoll-Filter

```
https://<IP-
Adresse|Domäne>:5000/specto/netflow?protocol=UDP&time=custom&start=01.10.
2020 17:19:20&end=13.12.2020 18:19:20
```

Listing 120. NetFlow Echtdaten-Abfrage mit Protokoll-Filter

Die Ausgaben werden nicht durch die Protokoll-Filterung beeinflusst und sehen damit in beiden Fällen folgendermaßen aus:

```
{
   "connections": 1000
}
```

Listing 121. JSON: NetFlow Echtdaten-Ausgabe

Statistiken

Die Abfrage der Statistiken kann auf zwei Arten erfolgen. Ist **`history`** für die Zeit angegeben, gibt es abermals den **`type`**-Parameter mit der Übergabe **`aktuell`**, um ausgehend vom aktuellen Datum die Statistiken abzufragen. Weiters kann mit der Übergabe **`custom`** und dem zusätzlichen Parameter **`moment=%d.%m.%Y`** die Abfrage der Statistiken von einem bestimmten Datum ausgehend erfolgen. Damit gibt es die folgenden zwei Abfrage-Formate:

```
https://<IP-Adresse|Domäne>:5000/specto/netflow?time=history&type=aktuell
```

Listing 122. NetFlow Statistiken-Abfragen type=aktuell

```
https://<IP-
Adresse|Domäne>:5000/specto/netflow?time=history&type=custom&moment=09.01
.2021
```

Listing 123. NetFlow Statistiken-Abfrage type=custom

Unabhängig vom angegebenen **`type`** werden immer die Anzahl der Verbindungen gesamt, pro Protokoll sowie pro Kategorie für jährlich, monatlich und täglich der ganzen Schule zurückgeliefert. Eine Ausgabe mit einem Ausschnitt der jährlichen Daten sieht somit wie folgt aus:

```
{
    "yearly": {
        "all": 5286412,
        "protocol": [
            {"TCP": 2341413},
            {"UDP": 2944999}
        ],
        "category": [
            {"Web": 3812451},
            {"File": 1473961}
        ]
    },
    "monthly": {…},
    "daily": {…}
}
```

Listing 124. JSON: Netflow Statistiken-Ausgabe

10.1.4 VPN

Für die Abfrage des VPN-Bereichs wird entsprechend **/vpn** angegeben. Dem **type**-Parameter können wie auch bereits bei SNMP die Werte **aktuell**, **custom** und **history** übergeben werden.

Echtdaten

Hier wird durch die Angabe von **aktuell** eine Liste der Source- und Destination-IP-Adressen der VPN Verbindungen und der zur öffentlichen IP-Adresse gehörigen Koordinaten der letzten Stunde zurückgeliefert. Selbiges gilt für **custom**, wo zusätzlich durch die Parameter **start=%d.%m.%Y %H:%M:%S** und **end=%d.%m.%Y %H:%M:%S** der Zeitraum für die gewünschten VPN-Verbindungen angegeben wird. Die Abfragen hierfür haben folgenden Aufbau und entsprechen auch dem Format der Abfragen der Bereiche „DNS-Tunneling“, „Externe DNS-Abfragen“ und „Login-Versuche“:

```
https://<IP-Adresse|Domäne>:5000/specto/vpn?time=aktuell
```

Listing 125. VPN Echtdaten-Abfrage type=aktuell

```
https://<IP-Adresse|Domäne>:5000/specto/vpn?time=custom&start=08.09.2020
11:00:00&end=09.09.2020 11:00:00
```

Listing 126. VPN Echtdaten-Abfrage type=custom

Wie bereits erwähnt besteht für beide Abfragen dasselbe Ausgabeformat, welches wie folgt aussieht und auch für die Echtdaten der Bereiche „DNS-Tunneling“, „Externe DNS-Abfragen“ und „Login-Versuche“ zutrifft:

```
[
    {
        "destination_ip": "8.8.8.8",
        "source_ip": "10.70.0.8",
        "latitude": 47.516231,
        "longitude": 14.550072
    },
    {
        "destination_ip": "8.8.8.8",
        "source_ip": "10.70.0.8",
        "latitude": 47.516231,
        "longitude": 14.550072
    }
]
```

Listing 127. JSON: VPN, DNS-Tunneling, externe DNS-Abfragen & Login-Versuche Echtdaten-Ausgabe

Statistiken

So wie auch bei „NetFlow“ gibt es den **type**-Parameter mit **aktuell** und **custom** und den zusätzlichen **moment**-Parameter, wenn **history** angegeben wird. Dadurch können die

statistischen Daten vom aktuellen Datum aus oder vom angegebenen Datum aus ausgegeben werden. Zusätzlich können immer die beiden optionalen Parameter **source=<IP-Adresse>** und **destination=<IP-Adresse>** angegeben werden. Anhand derer stehen die folgenden Abfrage-Kombinationen zur Verfügung:

- Angabe von keinem der beiden Parameter –> Summe der übertragenen Daten aller VPN-Verbindungen
- Angabe des **source**-Parameters –> Summe der übertragenen Daten von der übergebenen Source ausgehend
- Angabe des **destination**-Parameters –> Summe der übertragenen Daten an die übergebene Destination gehend
- Angabe des **source**- und **destination**-Parameters –> Summe der übertragenen Daten von der übergebenen Source an die übergebene Destination gehend

Wie bereits aus der Aufzählung hervorgeht, wird die Summe der übertragenen Daten immer abhängig von der gewählten Kombination zurückgeliefert. Die Anzahl der VPN-Verbindungen wird unabhängig von der Kombination pro Protokoll und pro Typ geliefert. Es werden immer die jährlichen, monatlichen und täglichen Statistiken zurückgeliefert. Entsprechend der Kombination kann eine Abfrage für VPNs wie folgt aussehen (das Format ist dasselbe für **aktuell** und **custom**):

```
https://<IP-
Adresse|Domäne>:5000/vpn?time=history&type=custom&moment=01.09.2020&sourc
e=10.70.0.8&destination=8.8.8.8
```

Listing 128. VPN mögliche Statistiken-Abfrage

Die Ausgabe hat unabhängig von der Kombination immer dasselbe Format und kann beispielsweise so aussehen:

```
{
    "yearly": {
        "sum": 31296688.0,
        "protocol": [
            {"UDP": 134513},
            {"TCP": 214153}
        ],
        "type": [
            {"OpenVPN[TCP]": 223154},
            {"WireGuard": 57291},
            {"OpenVPN[UDP]": 68221}
        ]
    },
```

```
    "monthly": {…},
    "daily": {…}
}
```

Listing 129. JSON: VPN Statistiken-Ausgabe

10.1.5 Portscan

Für den Portscan-Bereich sind für die Visualisierung nur die Statistiken relevant. Es gibt aber dennoch die beiden Werte **`aktuell`** und **`custom`** für den **`time`**-Parameter, da Team Oculus IP-Adressen für die Abfrage der statistischen Datensätze benötigt. Um die Abfrage dieses Bereichs durchzuführen, wird **`/portscan`** angegeben.

Statistiken

Somit gilt hier für **`aktuell`** und **`custom`** dasselbe Format für die Abfragen wie auch im Kapitel „VPN – Echtdaten“ definiert. In den Ausgaben werden lediglich die Destination- und Source-IP-Adresse zurückgeliefert. Die Ausgabe sieht also wie folgt aus:

```
[
   {
   "destination_ip": "10.0.0.254",
   "source_ip": "10.0.0.2",
   }
]
```

Listing 130. JSON: Portscan IP-Ausgabe

Um die tatsächlichen Statistiken abzufragen, wird **`type=history`** angegeben. Hierzu stehen dieselben Parameter zur Verfügung, wie sie für den Bereich „VPN“ definiert worden sind. Im Unterschied zu den VPN-Verbindungen wird abhängig von den IP-Parameter-Kombinationen jeweils nur die in der Datenbank gespeicherte Anzahl an Portscans zurückgeliefert. Eine Ausgabe hierzu sieht wie folgt aus:

```
{
    "yearly": {"count": 58323},
    "monthly": {…},
    "daily": {…}
}
```

Listing 131. JSON: Portscan Statistiken-Ausgabe

10.1.6 DNS-Tunneling

Der Bereich für das DNS-Tunneling wird über **`/dns-tunneling`** abgefragt. Hier werden wieder **`aktuell`** sowie **`custom`** für die Echtdaten-Abfragen und **`history`** für die Statistiken-Abfragen angegeben.

Echtdaten

Der Aufbau der Abfragen und alle benötigten Parameter sowie die Ausgaben sind ident zu denen des „VPN“-Bereichs und können daher dem entsprechenden Kapitel entnommen werden. Es muss lediglich die Bereichs-Bezeichnung in der Abfrage geändert werden.

Statistiken

Für die Abfragen der Statistiken stehen wieder die Parameter **`type`**, **`source`** und **`destination`** zur Verfügung. Die genaueren Definitionen und die Abfragemöglichkeiten können daher aus dem Bereich „VPN“ entnommen werden. Im Fall von DNS-Tunneling werden immer die Summe der übertragenen Daten abhängig von der gewählten Kombination und die Anzahl der DNS-Tunneling Verbindungen unabhängig von der Kombination zurückgeliefert. Damit hat eine Ausgabe folgenden Aufbau:

```
{
    "yearly": {
        "sum": 93653203.536,
        "count": 346232
    },
    "monthly": {…},
    "daily": {…}
}
```

Listing 132. JSON: DNS-Tunneling Statistiken-Ausgabe

10.1.7 Externe DNS-Abfragen

Um Informationen über externe abgefragte DNS-Server zu gewinnen, kann dies über **`/external-dns`** abgefragt werden. Auch hier wird über den **`time`**-Parameter zwischen den Abfragen von Echtdaten und den Abfragen der Statistiken unterschieden.

Echtdaten

Wie auch beim DNS-Tunneling Bereich entspricht der Aufbau der Echtdaten-Abfragen und Echtdaten-Ausgaben dem des „VPN“-Bereichs. Somit können nähere Informationen hierzu dem Kapitel „VPN – Echtdaten“ entnommen werden.

Statistiken

Grundsätzlich entspricht der Aufbau der Abfragen jenen des im „VPN“-Kapitel definierten Aufbaus. Allerdings stehen für die **`source`**- und **`destination`**-Parameter nur zwei Kombinationen zur Verfügung:

- Angabe des **`destination`**-Parameters –> Anzahl der Verbindungen zwischen der Schule und dem externen DNS-Server mit der angegeben IP-Adresse
- Angabe des **`destiantion`**- und **`source`**-Parameters –> Anzahl der Verbindungen zwischen der angegebenen Source und der angegeben Destination

Somit wird in jedem Fall der **`destination`**-Parameter benötigt. Es wird die Anzahl der Verbindungen abhängig von einer der beiden gewählten Kombinationen zurückgeliefert. Die Ausgabe hierzu sieht folgendermaßen aus:

```
{
    "yearly": {"count": 68362},
    "monthly": {…},
    "daily": {…}
}
```

Listing 133. JSON: externe DNS-Abfragen Statistiken-Ausgabe

10.1.8 Login-Versuche

Der Bereich „Login-Versuch“ wird über **`/login`** abgefragt. Für die interaktive Visualisierung werden hierzu von Team Oculus die Echtdaten benötigt und für Grafiken die Statistiken.

Echtdaten

Damit die Echtdaten abgefragt werden können, wird für den **`time`**-Parameter erneut **`aktuell`** oder **`custom`** mit dem zusätzlichen **`moment`**-Parameter angegeben. Die Abfragen und Ausgaben hierzu entsprechen dem in Kapitel „VPN“ definierten Format.

Statistiken

Im Fall der Statistiken wird der **`type`**-Parameter angegeben, um den Zeitpunkt der Abfrage festzulegen. Über die **`source`**- und **`destination`**-Parameter wird anhand der vier definierten Kombinationen, die im „VPN“-Bereich zu finden sind, bestimmt, worauf sich die Ausgabe bezieht. Je nach dem, welche Kombination gewählt wurde, wird die Anzahl der erfolgreichen und misslungenen Login-Versuche zurückgeliefert. Die Ausgabe der statischen Login-Versuche Abfrage sieht somit folgendermaßen aus:

```
{
    "yearly": {
        "count_success": 2319,
        "count_fail": 2183662
    },
    "monthly": {…},
    "daily": {…}
}
```

Listing 134. JSON: Login-Versuche Statistiken-Ausgabe

10.1.9 Programme

Im Bereich „Programme“ werden lediglich die Statistiken für die Visualisierung benötigt. Da aber so wie im „Portscan“-Bereich für die Abfrage-Kombinationen IP-Adressen angegeben werden müssen, gibt es hierzu dennoch die Optionen **`aktuell`** und **`custom`** für den **`time`**-Parameter. Für die Abfragen dieses Bereichs wird **`/program`** angegeben.

Statistiken

Wie bereits erwähnt, werden IP-Adressen für die Statistiken benötigt. Die entsprechenden Abfrage-Möglichkeiten und Ausgaben können dem Kapitel „Portscan“ entnommen werden. Es muss lediglich die andere Bereichsbezeichnung in den Abfragen eingefügt werden.

Damit nun die Statistiken selbst abgefragt werden können, stehen dieselben Parameter, wie im Kapitel „VPN“ definiert, zur Verfügung. Anhand der gewählten IP-Adressen-Kombination wird immer die Summe der Daten, die von den Programmen übertragenen worden sind, zurückgeliefert. Ebenso wird die Anzahl der Programm-Verbindungen unabhängig von der gewählten Kombination pro Typ ausgegeben. Auch hier werden immer die jährlichen, monatlichen und täglichen Statistiken zurückgeliefert. Die Ausgabe dieses Bereichs hat folgenden Aufbau:

```
{
    "yearly": {
        "sum": 3000.0,
        "type": [
            {"OneDrive": 6},
            {"YouTube": 12}
        ]
    },
    "monthly": {…},
    "daily": {…}
}
```

Listing 135. JSON: Programme Statistiken-Ausgabe

10.1.10 Rogue-DHCP-Server

Um die Rogue-DHCP-Server Daten abzufragen, wird **`/rogue-dhcp`** angegeben. Hierbei werden für die Visualisierung nur die Statistiken der ganzen Schule benötigt. Jedoch werden auch hierfür IP-Adressen benötigt, weshalb die Optionen **`aktuell`** und **`custom`** für den **`time`**-Parameter Anwendung finden.

Statistiken

Die Abfragen der IP-Adressen haben dasselbe Format wie im Kapitel „Portscan“ definiert. Der einzige Unterschied liegt in der Ausgabe, da es für diesen Bereich nur eine einzige IP-Adresse, die IP der verwendeten Rogue-DHCP-Server im Schulnetzwerk, gibt. Eine Ausgabe hierzu kann somit wie folgt aussehen:

```
[
    {"ip": "10.0.0.2"}
    {"ip": "10.0.0.3"},
]
```

Listing 136. JSON: Rogue-DHCP-Server IP-Adressen-Ausgabe

Um die tatsächlichen Statistiken zu bekommen, wird für den **`time`**-Parameter **`history`** angegeben. Es muss zusätzlich der **`type`**-Parameter mit der Option **`aktuell`** oder **`custom`** mit dem zusätzlichen **`moment`**-Parameter angegeben werden, um das Abfrage-Datum anzugeben. Zusätzlich gibt es den Parameter **`ip=<IP-Adresse>`**, über den zwei Kombinationen zur Verfügung stehen:

- Keine Angabe von **`ip=`** –> die Gesamtanzahl die in der Datenbank gespeicherten Rogue-DHCP-Server
- Angabe von **`ip=`** –> die Anzahl der in der Datenbank gespeicherten Rogue DHCP-Server mit der angegebenen IP-Adresse

Eine Abfrage mit Angabe von **`ip=`** kann folgendermaßen aussehen:

```
https://<IP-Adresse|Domäne>:5000/specto/rogue-
dhcp?time=history&type=custom&moment=01.09.2020&ip=10.70.0.8
```

Listing 137. Rogue-DHCP-Server Statistiken-Abfrage

Somit werden bei jeder Abfrage die Anzahl der Rogue-DHCP-Server abhängig von der Angabe des Parameters ausgegeben. Die Ausgabe hat also folgendes Format:

```
{
    "yearly": {"count": 2846},
    "monthly": {…},
    "daily": {…}
}
```

Listing 138. JSON: Rogue-DHCP-Server Statistiken-Ausgabe

10.1.11 IPs

Zuletzt können Statistiken zu IP-Adressen abgefragt werden. Dieser Bereich wird in der Abfrage mit **`/ip`** angegeben. Da hierfür jede IP-Adresse aus einem beliebigen Bereich verwendet werden kann, fallen die Optionen **`aktuell`** und **`custom`** für den **`time`**-Parameter weg.

Statistiken

Aus dem oben genannten Grund werden die Statistiken unter Angabe von **`time=history`** abgefragt. Dazu muss der **`type`**-Parameter entweder mit **`aktuell`** oder **`custom`** und dem zusätzlichen **`moment=%d.%m.%Y`**-Parameter angegeben werden. Ebenso gibt es zwei Parameter, um IP-Adressen anzugeben, und zwar **`public=<IP-Adresse>`** und

private=<IP-Adresse>. Aus diesen ergeben sich die folgenden zwei Parameter-Kombinationen:

- Angabe des **public**-Parameters –> Anzahl der Aufrufe der angegebenen Public-IP-Adresse
- Angabe des **public**- und **private**-Parameters –> Anzahl der Aufrufe der angegebenen Public-IP-Adresse ausgehend von der angegebenen Private-IP-Adresse

Damit kann die Abfrage dieses Bereichs beispielsweise so aussehen:

```
https://<IP-
Adresse|Domäne>:5000/specto/ip?time=history&type=custom&moment=01.09.2020
&private=10.70.0.8&public=8.8.8.8
```

Listing 139. IPs Statistiken-Abfrage

In der Ausgabe werden abhängig von der Kombination die Anzahl der IP-Adressen-Aufrufe ausgegeben. Da das Format jenem der Statistiken des Rogue DHCP-Server Bereichs entspricht, kann der Aufbau der Ausgabe dem Kapitel „Rogue-DHCP-Server“ entnommen werden.

10.2 Umsetzung mit Python

Nachdem der Aufbau der API festgelegt worden ist, wurde sie der Planung entsprechend programmiert[19]. Dies ist mit der Programmiersprache Python und insbesondere der Python-Library „Flask“ durchgeführt worden. Die Library ermöglicht das Erstellen einer **app**, also der eigentlichen API, für welche beispielsweise bestimmte Konfigurationen, API-Pfade oder auch Error-Handler angegeben werden können. Der Import von Flask wurde folgendermaßen durchgeführt:

```
from flask import Flask
```

Listing 140. Python: Flask-Import

Ebenso wichtig für die Umsetzung ist der Zugriff auf die beiden Datenbanken. Für die Echtdaten-Datenbank, bei der es sich um MongoDB handelt, werden die Abfragen mit Hilfe der Python-Library „PyMongo“ durchgeführt. Bei der Statistik-Datenbank handelt es sich um MySQL-Server. Diese Datenbank wird über die „MySQL-Connector“-Library abgefragt. Genauere Informationen zur Einbindung können den Kapiteln „MongoDB mit Python“ und „MySQL mit Python“ entnommen werden.

Nach der Umsetzung der Abfragen sind ergänzende Funktionen wie eine Möglichkeit zur Authentifizierung zum Error-Handling und zum Logging ergänzt worden. Nähere Informationen diesbezüglich kann den nachfolgenden Unterkapiteln entnommen werden.

[19] Das vollständige Skript ist als externe Datei „Specto_API.py“ auf der CD des Bibliotheksexemplars hinterlegt.

10.2.1 Abfragen

Bevor die Abfragen durchgeführt werden konnten, musste zunächst die **app** erstellt werden. Dies wurde wie folgt umgesetzt:

```
app = Flask(__name__)
```

Listing 141. Python: Erstellung der app

Daraufhin ist zu jedem zuvor in der Planung definierten Bereich eine eigene Funktion erstellt worden. Vor jeder dieser Funktionen ist der Pfad für den Aufruf und die verwendete http-Methode angegeben. Im Pfad ist jeweils **/specto** der Root-Knoten, auf welchen die jeweiligen Bereichsbezeichnung im Format **/<Bereichsname>** folgt. Bei der http-Methode handelt es sich bei allen Bereichen um die Get-Methode. Am Beispiel des Klassenbereichs sieht dies nun wie folgt aus:

```
@app.route('/specto/classes', methods=['GET'])
def classroom():
```

Listing 142. Python: Funktionsdefinition des Klassen-Bereichs

Im Großen und Ganzen ist die Vorgehensweise für alle Bereichen jeweils für die Echtdaten und für die Statistiken sehr ähnlich und unterscheidet sich nur anhand von Änderungen an Selects oder anhand anderer **if**-Statements. Zur Unterscheidung zwischen der benötigten Datenstruktur wird im ersten Schritt in jeder Bereichs-Funktion eine Variable für den „time"-Parameter benötigt. Anhand dieser wird anschließend mittels **if**-Statement der benötigte Code-Bereich bestimmt. Die Speicherung des übergebenen Parameters wird folgendermaßen durchgeführt:

```
timestatus = str(request.args["time"])
```

Listing 143. Python: "time"-Parameter in Variable speichern

Im Folgenden werden die Vorgehensweisen jeweils für die Echtdaten und für die Statistiken allgemein erläutert und anhand von Beispielen aufgezeigt.

Echtdaten

Da die Echtdaten aus datenschutzrechtlichen Gründen nicht in Echtzeit analysiert werden dürfen, gelangen sie erst mit einer statisch gesetzten Verzögerung in die Verarbeitung und daraufhin in die Echtdaten-Datenbank. Damit die Daten über die API aber dennoch **aktuell**, das heißt als wäre es der aktuelle Tag, abgefragt werden können, ist in der API die Verzögerung über eine globale Variable definiert. Anhand dieser werden aktuelle Zeitabfragen angepasst.

Um die Abfragen der Echtdaten durchführen zu können, muss im ersten Schritt eine Verbindung mit der Echtdaten-Datenbank aufgebaut werden. Dies erfolgt global und wird wie folgt durchgeführt:

```
client_specto =
MongoClient("mongodb://specto:gansgeheim123!@10.70.8.1:27017/")
db_specto = client_specto["specto"]
```

Listing 144. Python: Verbindung zur Echtdaten-Datenbank

In jedem Bereich kann daraufhin über das Format **`db_specto["<Table-Name>"]`** auf die jeweiligen Bereichs-Datensätze zugegriffen werden. Im Fall von SNMP sieht dies so aus:

```
col_snmp_entries = db_specto["snmpentries"]
```

Listing 145. Python: Zugriff auf die SNMP-Entries

Anschließend wird je nach Übergabe für den **`time`**-Parameter der Abfrage-Zeitraum bestimmt, um die benötigte Abfrage durchführen zu können. Sofern es **`aktuell`** als Wert für einen Bereich gibt, wird mit Hilfe der Verzögerung der jeweilige in der Planung definierte Zeitraum angegeben. Für den VPN-Bereich wird dies folgendermaßen umgesetzt:

```
endtime = datetime.datetime.today() - datetime.timedelta(days=time_diff)
starttime = endtime - datetime.timedelta(hours=1)
```

Listing 146. Python: VPN-Bereich Echtdaten "aktuell"-Zeitraum-Bestimmung

Bei **`custom`** wird mit den übergebenen Parametern der Zeitraum definiert. Dazu werden zunächst entsprechende Variablen angelegt, für welche anschließend das zur Übergabe benötigte Zeitformat angegeben wird. Am Beispiel des VPN-Bereichs sieht dies wie folgt aus:

```
time_format = "%d.%m.%Y %H:%M:%S"
starttime = datetime.datetime.strptime(start, time_format)
endtime = datetime.datetime.strptime(end, time_format)
```

Listing 147. Python: VPN-Bereich Echtdaten "custom"-Zeitraum-Bestimmung

Generell wird eine Abfrage jeweils über **`col_<Bereich>_entries.find()`** durchgeführt. In der runden Klammer wird in geschwungenen Klammern das gesuchte Feld angegeben, in welchem die Datensätze herausgefiltert werden, die in den gewünschten Zeitraum fallen. Am Beispiel von VPN sieht dies folgendermaßen aus:

```
col_vpn_entries.find({'stop': {'$gte': starttime, '$lt': endtime}})
```

Listing 148. Python: Echtdaten VPN-Entries eingrenzen

Werden IP-Adressen übergeben und in der Ausgabe benötigt, so werden diese über eigene Funktionen zu einer Integer-Zahl umgewandelt beziehungsweise auch von einer *Integer*-Zahl zu einem String für die Ausgabe zurückgewandelt. Dies ist notwendig, da sie als *Integer* in den Datenbanken abgespeichert sind. Bei Bereichen, die IP-Adressen benötigen, muss zusätzlich auf die IP-Datensätze und die Länder-Datensätze zugegriffen werden, damit die Koordinaten zu öffentlichen Adressen ausgegeben werden können. Sind die Datensätze eingegrenzt, wird für jeden Datensatz über **`entry["<Feldbezeichnung>"]`** der jeweils

benötigte Wert ausgelesen. Die Werte werden mit geschwungenen Klammern im definierten Ausgabeformat geschrieben und letztendlich mittels der „jsonify“-Library als JSON zurückgeliefert. Am Beispiel des Klassen-Bereichs sieht dies folgendermaßen aus:

```
for entry in entries:
   data = {"nr": entry["_id"], "description": entry["description"]}
   classes.append(data)
return jsonify(classes)
```

Listing 149. Python: Klassen - Ausgabe der Daten

Statistiken

Um die Statistiken abzufragen, muss für den **`time`**-Parameter der Wert **`history`** angegeben werden. Wie auch bei den Echtdaten, muss im ersten Schritt eine Verbindung mit der Statistik-Datenbank hergestellt werden. Der Verbindungsaufbau erfolgt mit Hilfe der „mysql-connector“-Library. Anders als bei den Echtdaten wird die Verbindung nicht global aufgebaut, sondern pro Bereich. Des Weiteren wird jeweils ein Cursor erstellt, über den anschließend die Selects durchgeführt werden. Im Fall von NetFlow erfolgt der Verbindungsaufbau wie folgt:

```
db_connections = mysql.connector.connect(host="10.70.8.1",
user="specto",password="gansgeheim123!", database="connections")
connections_cursor = db_connections.cursor(buffered=True)
```

Listing 150. Python: NetFlow Verbindung zur Statistik-Datenbank

Anschließend ist der für den Bereich benötigte Select definiert worden, dieser wird einmal in einer Variable definiert und kann daraufhin, wenn er benötigt wird, ausgeführt werden. Für den NetFlow-Bereich sieht der Select so aus:

```
select = "select protocol, type, amount from facts
   inner join connection on fk_connection_id = pk_connection_id
   inner join time on fk_time_id = pk_time_id
   inner join ip s on fk_source_ip = s.pk_ip
   inner join ip d on fk_destination_ip = d.pk_ip
   where name = %s and year = %s and month = %s and day = %s and
   fk_source_ip = %s and fk_destination_ip = %s"
```

Listing 151. Python: NetFlow Statistiken-Select

Bevor die gewünschten Daten aufbereitet werden können, muss auch für die Statistiken die Abfrage-Zeit definiert sein. Hat hier der **`type`**-Parameter den Wert **`aktuell`**, wird mit der statischen Verzögerung vom aktuellen Datum ausgegangen. Dies ist folgendermaßen umgesetzt worden:

```
clock = datetime.date.today() - datetime.timedelta(days=time_diff)
```

Listing 152.Python: Statistiken "aktuell"-Datum

Bei der Übergabe von **custom** wird aus dem **moment**-Parameter das übergebene Datum ausgelesen. Daraufhin wird es im zur Übergabe benötigten Format in einer Variablen gespeichert. Dies wird in der API so durchgeführt:

```
moment = str(request.args['moment'])
clock = datetime.datetime.strptime(moment, "%d.%m.%Y")
```

Listing 153. Python: Statistiken "custom"-Datum

Mit Hilfe der **clock**-Variable wird nun jeweils der entsprechende Select ausgeführt. Dabei sind im Select Felder, die mit übergebenen Werten beim Aufruf gefüllt werden sollen, mittels **%s** gekennzeichnet. Diese sind besonders relevant, um die jährlichen, monatlichen und täglichen Statistiken abzufragen. So wird für die jährlichen Statistiken lediglich das Jahr der **clock**-Variable übergeben, bei den monatlichen Statistiken das Jahr sowie das Monat und bei den täglichen Statistiken das Jahr, das Monat sowie der Tag. Über **fetchall()** werden anschließend alle zutreffenden Datensätze in eine Variable gespeichert. Am Beispiel von NetFlow ist das folgendermaßen umgesetzt worden:

```
connections_cursor.execute(select, ("NetFlow", clock.year, 0, 0, 0, 0))
year = connections_cursor.fetchall()
connections_cursor.execute(select, ("NetFlow", clock.year, clock.month,
0, 0, 0))
month = connections_cursor.fetchall()
connections_cursor.execute(select, ("NetFlow", clock.year, clock.month,
clock.day, 0, 0))
day = connections_cursor.fetchall()
```

Listing 154. Python: NetFlow Statistiken: Ausführen des Selects

Um an die für die Ausgabe benötigten Daten zu gelangen, iteriert eine **foreach**-Schleife jeweils über die in den Variablen gespeicherten Datensätze und über **[<Feldnummer>]** wird auf die im Select bestimmten Rückgabefelder zugegriffen. Diese werden im für die Ausgabe benötigten Format in Listen mitgespeichert und letztendlich in der Ausgabe entsprechend eingefügt. Damit die Daten auch als *JSON* geliefert werden, wird auch hier die „jsonify"-Library benötigt. Im Fall von NetFlow ist das **return**-Statement somit so aufgebaut:

```
return jsonify(
{"yearly": {"all": year_all, "protocol": year_protocol, "category":
year_type},
"monthly": {"all": month_all, "protocol": month_protocol, "category":
month_type},
"daily": {"all": day_all, "protocol": day_protocol, "category":
day_type}})
```

Listing 155. Python: NetFlow Statistiken return-Statement

10.2.2 Authentifizierung

Die API soll nur von Team „Oculus“ für die einprogrammierten Abfragen, die für die Visualisierung benötigt werden, genutzt werden können. Daher ist eine Authentifizierung in Form von *Cookies* implementiert worden. Dies ist mit Hilfe der Library „flask_jwt_extended“ umgesetzt worden. Im ersten Schritt sind entsprechende Konfiguration an der **app** durchgeführt worden. Besonders das Konfigurationsfeld „JWT_SECRET_KEY“ ist relevant, da anhand dessen das *Cookie* generiert wird. Die Anpassungen im Code sehen wie folgt aus:

```
app.config['JWT_TOKEN_LOCATION'] = ['cookies']
app.config['JWT_ACCESS_COOKIE_PATH'] = '/specto/'
app.config['JWT_COOKIE_CSRF_PROTECT'] = True
app.config['JWT_SECRET_KEY'] = '<Zeichenkette>'
```

Listing 156. Python: app.config für die Cookies

Damit man das für den Zugriff benötigte Cookie bekommt, ist eine Funktion ergänzt worden, die über folgenden Pfad aufgerufen wird:

```
https://<IP-Adresse|Domäne>:5000/token/login
```

Listing 157. Login-Pfad

Ein wichtiger Unterschied im Vergleich zu den anderen Abfragen ist, dass es sich um eine POST-Abfrage handelt. Daher müssen die entsprechenden Logindaten, Username und Passwort, als *JSON-Daten* im Body des Requests mitgeliefert werden. Der Body muss damit so aussehen:

```
{
   "username": "Specto",
   "password": "gansgeheim123!"
}
```

Listing 158. JSON: Login Request-Body

Aufgrund der Verwendung einer Authentifizierung wird aus Sicherheitsgründen der Zugriff mittels *Javascript* wegen möglichem *Cross Site Scripting* nicht zugelassen. Diesbezüglich mussten noch die folgenden Änderungen an der **app**-Konfiguration vorgenommen werden sowie die *CORS-Policy* eingestellt werden:

```
app.config['JWT_COOKIE_SECURE'] = True
app.config['JWT_COOKIE_SAMESITE'] = "None"
CORS(app, supports_credentials=True)
```

Listing 159. Python: Anpassungen für den API-Zugriff von Team Oculus

Zuletzt ist eine sichere Verbindung über *https* erforderlich. Dafür werden von „LetsEncrypt“ ausgestellte Zertifikate verwendet. Die genaue Vorgehensweise für die Umsetzung mit https kann dem Kapitel „Implementierung der API“ entnommen werden.

10.2.3 Error-Handling

Im Fehlerfall, durch zum Beispiel falsche Abfragen, soll keine *HTML-Seite* mit der Fehlernachricht erscheinen, sondern der Fehler im *JSON-Format* ausgegeben werden. Hierzu wurde die in Flask integrierte Funktion von Error-Handlern angewandt. Über diese kann für ausgewählte Fehlercodes die Ausgabe manuell angepasst werden. In diesem Fall ist dies für die Fehlercodes 404 (falsche Eingaben) und 405 (nicht erlaubte http-Methoden) umgesetzt worden.

Grundsätzlich sind von Team Oculus alle Abfragen statisch einprogrammiert worden, wodurch im Prinzip die Notwendigkeit für angepasste Fehlernachrichten für alle möglichen Fehlercodes nicht gegeben ist. Da es aber im Prozess der Erstellung immer wieder zu Falscheingaben kommen kann und es für Team Oculus einfacher ist, direkt einen Fehler in JSON einzulesen, wurden die zwei zuvor erwähnten *Error-Handler* für die API programmiert.

Wie bereits bei den API-Pfaden, gibt es für den *Error-Handler* eine entsprechende Programmzeile im Format **`@app.errorhandler(<Error-Code>)`**. Auf diese folgt eine entsprechende Funktion mit einem Parameter, dem Fehler. In dieser wird daraufhin lediglich eine entsprechende Nachricht in JSON und der Fehlercode selbst zurückgeliefert. Am Beispiel des 404-Error-Codes sieht dies wie folgt aus:

```
@app.errorhandler(404)
def notfound(e):
    return jsonify({"error 404": "Please enter a valid URL"}), 404
```

Listing 160. Python: Error-Handler 404

10.2.4 Logging

Um im Nachhinein besser die Verwendung der API nachvollziehen zu können und etwaige Fehler leichter ausfindig zu machen, wurde das Speichern von Log-Nachrichten in einem eigenen Log-File implementiert. Dazu werden die in Flask integrierten Debug-Nachrichten, die im Debug-Modus in der Konsole ausgegeben werden, umformatiert und in ein File gespeichert. Dazu musste die „logging“-Library importiert werden und die folgende Ergänzung am Code vorgenommen werden:

```
import logging
logging.basicConfig(filename='C:/Users/Specto/Desktop/Scripts/log/log.txt
',level=logging.DEBUG,
format='%(asctime)s\t%(levelname)s\t%(threadName)s\t: %(message)s')
```

Listing 161. Python: Anpassungen für das Logging

Die Log-Zeilen werden damit nach dem folgenden Format aufgebaut:

```
Timestamp Debug-Level Thread-Name : Debug-Message (IP-Adresse - -
Timestamp Type Abfrage Status-Code)
```

Listing 162. Log-Zeilen Format

Eine tatsächliche Log-Zeile sieht damit zum Beispiel so aus:

```
2020-12-18 20:48:15,012 INFO Thread-2 : 10.70.8.253 - - [18/Dec/2020
20:48:15] "[37mGET /specto/classes HTTP/1.1[0m" 200 –
```

Listing 163. Beispiel einer Log-Zeile

11 DMZ-Host

Beim DMZ-Host handelt es sich um eine sich in einer DMZ der Schule befindlichen Maschine. Auf dieser wird die Echtdaten- sowie die Statistik-Datenbank gehostet. Schließlich muss die Medientechnik, deren Website auf einem schulexternen Webserver gehostet wird, Zugriff auf die API und damit auf die beiden Datenbanken haben.

Bei einer Demilitarisierten Zone (DMZ) handelt es sich um eine Art Pufferzone zwischen einem externen und internen Netzwerk. Die einzelnen Zonen – LAN, WAN und DMZ – sind durch Firewalls voneinander getrennt. In einer DMZ befinden sich Server, Webserver und Mailserver, welche für Nutzer aus dem Internet erreichbar sind. Ein Zugriff aus der DMZ ins LAN und somit auf interne Ressourcen ist nicht möglich. (vgl. Luber 2018)

Wendet man dieses Wissen auf die Situation dieser Diplomarbeit an, geht das Erfordernis dieses Hosts besser hervor. Die API muss in jedem Fall aus dem Internet erreichbar sein. Deshalb müssen die Datenbanken ebenfalls auf dieser Maschine gehostet werden, da die API ansonsten nicht auf die Ressourcen innerhalb des Schulnetzwerkes und damit auf die Datenbanken zugreifen könnte. Der Zugriff aus dem Schulnetzwerk ist jedoch möglich und somit können die Daten in die Datenbank mittels der Datenbank-Skripte beschrieben werden. Geht man davon aus, dass der DMZ-Host von einem Angreifer kompromittiert wird, kann dieser nicht weiter in das Schulnetzwerk eindringen und keinen Schaden anrichten.

Die folgende Abbildung dient zum besseren Verständnis:

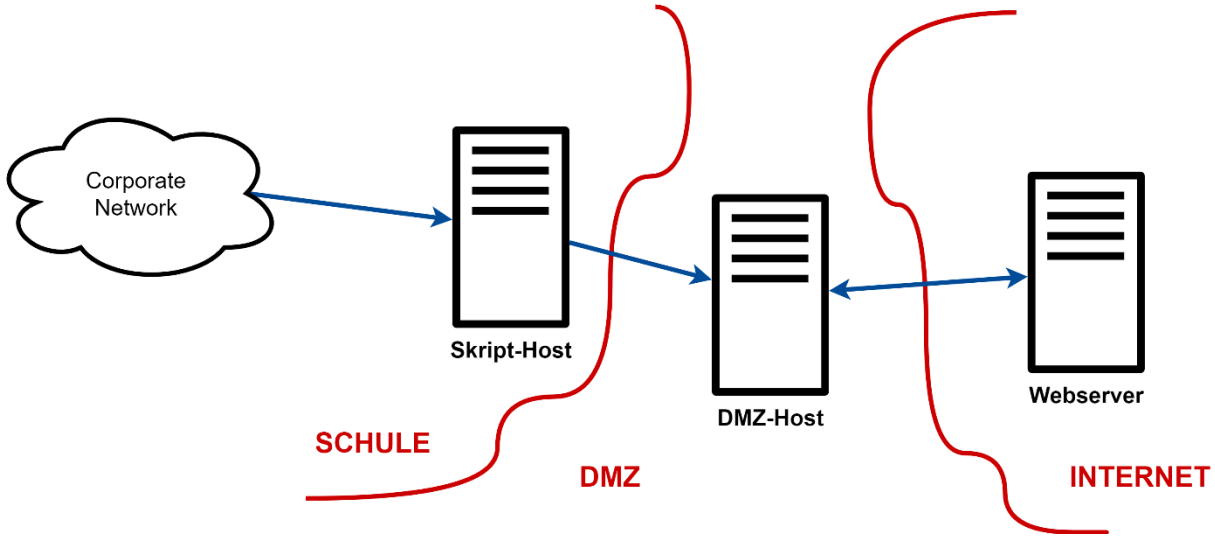


Abbildung 37. Darstellung der erlaubten Zugriffe auf den DMZ-Host

Die blauen Pfeile veranschaulichen die möglichen Zugriffsrichtungen und die roten Linien grenzen die einzelnen Zonen ein. Auf dem Skript-Host laufen die Datenbank-Skripte zum Beschreiben der beiden Datenbanken. Der Webserver stellt den Server der Medientechnik dar, welcher Abfragen durch die API ausführt, während die API entsprechend antwortet.

11.1 Ubuntu Server

Als Betriebssystem für den DMZ-Host wird die Linux-Distribution Ubuntu-Server verwendet. Somit handelt es sich dabei um ein frei verwendbares Betriebssystem. Außerdem ist es eines der weitverbreitetsten im Serverbereich. Es ist zuverlässig, läuft stabil und ist sehr performant.

11.2 Aufsetzen der Datenbanken

Wie bereits oben erwähnt, werden auf dem DMZ-Host die Datenbanken gehostet. Bei der Echtdaten-Datenbank handelt es sich um den MongoDB Community Server 4.4. Der MySQL Server 8.0 wird als Statistik-Datenbank verwendet. Diese beiden Datenbanken sind im Netzwerk über eine Authentifizierung erreichbar gemacht worden. Weiters wurden entsprechende *Collections* und Tabellen eingerichtet[20].

11.2.1 MongoDB

Nach dem Installieren von MongoDB auf dem DMZ-Host mittels dem Paketverwaltungssystem APT [21], ist für die Datenbank keine Authentifizierung eingerichtet. Aufgrund dessen wurde ein entsprechender User für die Diplomarbeit über eine Instanz ohne Authentifizierung angelegt. Nach dem Anlegen wurde in der Konfigurationsdatei die Authentifizierung verbindlich eingerichtet. Weiters musste auch für die Erreichbarkeit der Datenbank im Netzwerk die an die MongoDB gebundene IP-Adresse in der Konfiguration geändert werden. MongoDB läuft nun als Service auf dem DMZ-Host. (vgl. MongoDB o. J. d)

Bevor die Datenbank beschrieben werden kann, wurden die aus der Datenbankplanung hervorgehenden *Collections* für die Datenbank Specto erstellt. Diese können über die Mongo-Shell angelegt und verwaltet werden. Eine *Collection* wird mittels folgender Zeile erstellt:

```
db.createCollection(<name>)
```

Listing 164. Mongo-Shell: Erstellen einer Collection

Weiters wurde die Echtdaten-Datenbank für die Abfrage von Dokumenten optimiert. Die einfachste Möglichkeit dies zu bewerkstelligen, ist die Erstellung von Indizes. Ein Index ermöglicht es, schneller nach geforderten Informationen zu suchen. Dabei handelt es sich um spezielle Datenstrukturen, welche einen kleinen Teil der Datensätze in leicht zu durchsuchender Form speichern. Der Wert eines Feldes wird also gespeichert und anhand dessen sortiert. Wird dies also auf ein Feld mit Zeitstempeln angewendet, werden diese danach sortiert, wie aktuell diese sind. (vgl. MongoDB o. J. e)

[20] Die vollständige Dokumentation zum Aufsetzen der „Maschine“ ist als externe Datei „Specto_Datenstruktur_DMZ-Host.pdf“ auf der CD des Bibliotheksexemplars hinterlegt.
[21] Advanced Packaging Tool

Einerseits ist es für die Erstellung von Statistiken in der Diplomarbeit relevant, schnellstmöglich alle noch nicht verarbeiteten Statistiken abzufragen. Andererseits ist es für die API der Diplomarbeit wichtig, Datensätze nach einem Zeitstempel abzufragen. Hierfür wurde in allen dafür erforderlichen *Collections* ein Index für das Feld **`processed`**, für das Feld **`selected`** sowie für diverse Zeitstempel erstellt. Für SNMP wurden über die Mongo-Shell mittels folgender Zeilen Indizes erstellt:

```
db.snmpentries.createIndex({processed: 1})
db.snmpentries.createIndex({selected: 1})
db.snmpentries.createIndex({timestamp: 1})
```

Listing 165. Mongo-Shell: Erstellung der SNMP-Indizes

Weiters wurden für die *Collections* **`classrooms`** und **`countries`** die Datensätze vordefiniert erstellt. Diese sind für die Verwendung der Datensätze durch die Medientechnik notwendig.

11.2.2 MySQL DB

Nach der Installation von MySQL-Server wird während des Ausführens des Sicherheit-Skripts ein User für die Diplomarbeit angelegt. Auch in der Konfigurationsdatei des MySQL-Servers muss die an diesen gebundene IP-Adresse für die Erreichbarkeit angepasst werden. Anschließend ist der Server bereit für das Anlegen der zwei separaten Statistik-Datenbanken. Diese werden anhand der beiden *Star-Schemen* (siehe Kapitel Star-Schemen) erstellt.

Neben der Erstellung der beiden Schemen, müssen auch vordefinierte Datensätze eingefügt werden. So müssen auch hier für die Statistik-Datenbank bezüglich SNMP-Einträge für die einzelnen Klassenräume eingefügt werden.
Für die IP-bezogene Statistik-Datenbank werden vordefinierte Connection-Datensätze erstellt. Diese sind ebenfalls für die Erstellung der Statistiken unbedingt vonnöten. Connection-Einträge bezüglich bestimmter Protokolle oder Typen werden variabel automatisiert eingetragen.

11.3 Implementierung der API

Wie bereits zuvor erwähnt, wird die Website von Team Oculus auf einem schulexternen Webserver gehostet. Damit die API somit erreichbar ist, muss sie öffentlich abrufbar sein und Zugriff auf die Echtdaten-Datenbank sowie die Statistik-Datenbank haben. Um dies zu ermöglichen, läuft die API gemeinsam mit den Datenbanken auf dem DMZ-Host. Entsprechend ist sie auf dem Host implementiert worden und es wurden über die Datei **`requirements.txt`** alle benötigten Python-Libraries installiert. Die Installation ist wie folgt durchgeführt worden:

```
python3 -m pip install -r requirements.txt
```

Listing 166. Linux-Terminal: API-Installation der benötigten Libraries

Daraufhin ist ein *crontab* für das automatische Starten der API nach einem Neustart erstellt worden. So wie bereits für SNMP wurde dazu ein entsprechender Eintrag, nach Eingabe von **`crontab -e`**, im Terminal erstellt. Dieser sieht folgendermaßen aus:

```
@reboot python3 /home/specto/API/Specto_API.py
```

Listing 167. „crontab"-Eintrag für das Starten der API

Damit ist die API bereits funktionsfähig, musste allerdings noch auf der Firewall auf eine öffentliche IP-Adresse übersetzt werden. Des Weiteren ist vom Administrator für die öffentliche IP der API ein *A-Record* erstellt worden. Dieser ist für die benötigte https-Funktion, die für den Zugriff von Team Oculus vonnöten ist, erforderlich.

11.3.1 Anpassungen für HTTPS

Um gültige Zertifikate für die API zu erhalten, wurde die Gratis-Möglichkeit von „Let's Encrypt" verwendet. Dies ist auch der Grund, weshalb ein *A-Record* für die API notwendig ist. Für die Umsetzung ist im ersten Schritt „Certbot" auf dem DMZ-Host installiert worden. Folgendermaßen wurde dies durchgeführt:

```
apt-get install certbot
```

Listing 168. Linux-Terminal: Installation von Certbot

Anschließend konnten bereits die benötigten Zertifikate erzeugt werden. Hierzu ist folgender Befehl verwendet worden:

```
certbot certonly --standalone --preferred http -d <A-Record>
```

Listing 169. Linux-Terminal: Zertifikat und private Key generieren

Die erzeugten Zertifikate befinden sich im Verzeichnis **`/etc/letsencrypt/live/<A-Record>`** und müssen für funktionierendes *https* nur noch in der API angegeben werden. Dazu muss im Aufruf der API der **`ssl_context`** angegeben werden, welchem die beiden Files **`fullchain.pem`** und **`privkey.pem`** übergeben werden. Die Datei **`fullchain.pem`** beinhaltet das eigene sowie alle darüberliegenden Zertifikate, während

sich in dem File **`privkey.pem`** der private Key der API zum Entschlüsseln des verschlüsselten Netzwerkverkehrs befindet. Dies wurde folgendermaßen umgesetzt:

```
app.run(debug=True, host='0.0.0.0', ssl_context=(
'/etc/letsencrypt/live/<A-Record>/fullchain.pem',
'/etc/letsencrypt/live/<A-Record>/privkey.pem'))
```

Listing 170. Python: Angabe des Zertifikats und private Keys in der API

Daraufhin ist die API unter „https://“ mit einem gültigen Zertifikat erreichbar.

11.3.2 Automatische Erneuerung des Zertifikats

Da die Zertifikate von „Let’s Encrypt“ jedoch nur für drei Monate gültig sind, wurde noch eine automatische Erneuerung dieser implementiert. Dies ist mit Hilfe der Website „Certbot“ von *archlinux* umgesetzt worden. Entsprechend der Website wurde wie folgt ein Service für das Erneuern der Zertifikate erstellt (archlinux o.J.):

```
[Unit]
Description=Let's Encrypt renewal

[Service]
Type=oneshot
ExecStart=/usr/bin/certbot renew --quiet --agree-tos
```

Listing 171. Linux-Terminal: Service zum Erneuern des Zertifikats

Ebenso wurde ein weiterer Service erstellt, der zweimal am Tag auf eine mögliche Erneuerung prüft. Die entsprechende Service-Konfiguration sieht wie folgt aus (archlinux o.J.):

```
[Unit]
Description=Twice daily renewal of Let's Encrypt's certificates

[Timer]
OnCalendar=0/12:00:00
RandomizedDelaySec=1h
Persistent=true

[Install]
WantedBy=timers.target
```

Listing 172. Linux-Terminal: Service zum Überprüfen der Zertifikats-Erneuerung

Letztendlich ist der Service mittels „systemctl enable“ aktiviert worden und in der Datei **`/etc/letsencrypt/renewal/<A-Record>.conf`** wurde folgende Zeile für das automatische Neustarten des Hosts hinzugefügt:

```
renew_hook = /sbin/reboot
```

Listing 173. Linux-Terminal: Einstellen des Neustarts nach einer Zertifikats-Erneuerung

Damit ist die API fertig auf dem DMZ-Host implementiert und über https im Internet erreichbar. Weiters werden aufgrund der Services die Zertifikate automatisch erneuert, wodurch es zu keinem Ablauf des Zertifikats kommt.

Literaturverzeichnis

Administrator (2009): MySQL gleichzeitiger Zugriff, [online] https://administrator.de/forum/mysql-gleichzeitiger-zugriff-109110.html [Aufruf: 08.12.2020].

archlinux (o. J.): Certbot, [online] https://wiki.archlinux.org/index.php/Certbot#systemd [Aufruf: 28.02.2021].

Cisco (2011): NetFlow Version 9 Flow-Record Format, [online] https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html [Aufruf: 24.08.2020].

Cisco (o. J.): NetFlow Configuration Guide, Cisco IOS Release 15M&T, [online] https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/netflow/configuration/15-mt/nf-15-mt-book/get-start-cfg-nflow.html [Aufruf: 10.08.2020].

Datenbanken-verstehen (o. J.): Star-Schema – Sternschema, [online] https://www.datenbanken-verstehen.de/data-warehouse/data-warehouse-grundlagen/data-warehouse-datenmodell/star-schema/ [Aufruf: 24.11.2020].

DPS Telecom (o. J.): What is SNMPv1, SNMPv2c, and SNMPv3?, [online] https://www.dpstele.com/snmp/v1-v2c-v3-difference.php [Aufruf: 19.11.2020].

Drilling, Thomas (2017): Konzept, Aufbau und Funktionsweise von REST, [online] https://www.dev-insider.de/konzept-aufbau-und-funktionsweise-von-rest-a-603152/ [Aufruf: 02.11.2020].

Dörner, Jurek (2019): Der OLAP-Würfel, [online] https://data-science-blog.com/blog/2019/02/16/olap-wuerfel/ [Aufruf: 07.02.2021].

Fuchs, Elmar (2018): *SQL: Grundlagen und Datenbankdesign*, 5. Ausgabe, Bodenheim, Deutschland: HERDT-Verlag.

Google Developers (o. J.): countries.csv | Dataset Publishing Language | Google Developers, [online] https://developers.google.com/public-data/docs/canonical/countries_csv [Aufruf: 12.12.2020].

Hagel, Jens (o. J.): Was ist ein Portscan?, [online] https://www.hagel-it.de/it-service/was-ist-ein-portscan.html [Aufruf: 13.02.2021].

IANA (2021): Service Name and Transport Protocol Port Number Registry, [online] https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?&page=1 [Aufruf: 14.12.2020].

IONOS (2018): SNMP (Simple Network Management Protocol): Die Protokollbasis für Netzwerkmanagement, [online] https://www.ionos.at/digitalguide/server/knowhow/snmp-simple-network-management-protocol/ [Aufruf: 19.11.2020].

IONOS (2020a): ISO 8601 – Zeitangaben international sicher kommunizieren, [online] https://www.ionos.at/digitalguide/websites/web-entwicklung/iso-8601/ [Aufruf: 02.02.2021].

IONOS (2020b): APIs: Was man über die Programmierschnittstellen wissen muss, [online] https://www.ionos.at/digitalguide/websites/web-entwicklung/was-ist-eine-api/ [Aufruf: 02.11.2020].

IONOS (2020c): Ressourcenorientierte Webservices auf Basis von REST, [online] https://www.ionos.at/digitalguide/server/knowhow/rest-die-http-loesung-fuer-webservices/ [Aufruf: 02.11.2020].

Luber, Stefan (2017a): Was ist NoSQL?, [online] https://www.bigdata-insider.de/was-ist-nosql-a-615718/ [Aufruf: 02.02.2021].

Luber, Stefan (2017b): Was ist eine relationale Datenbank?, [online] https://www.bigdata-insider.de/was-ist-eine-relationale-datenbank-a-643028/ [Aufruf: 05.02.2021].

Luber, Stefan (2017c): Was ist MySQL?, [online] https://www.bigdata-insider.de/was-ist-mysql-a-614184/ [Aufruf: 05.02.2021].

Luber, Stefan (2017d): Was ist ein OLAP Cube?, [online] https://www.bigdata-insider.de/was-ist-ein-olap-cube-a-654603/ [Aufruf: 07.02.2021].

Luber, Stefan (2018): Was ist eine DMZ (Demilitarized Zone)?, [online] https://www.security-insider.de/was-ist-eine-dmz-demilitarized-zone-a-677267/ [Aufruf: 21.02.2021].

Luber, Stefan (o. J.): Was ist eine API?, [online] https://www.dev-insider.de/was-ist-eine-api-a-583923/ [Aufruf: 02.11.2020].

Manhart, Klaus (2008a): Business Intelligence (Teil 3): Datenmodellierung – Relationale und Multidimensionale Modelle, [online] https://www.tecchannel.de/a/business-intelligence-teil-3-datenmodellierung-relationale-und-multidimensionale-modelle,1744994 [Aufruf: 06.02.2021].

Manhart, Klaus (2008b): Business Intelligence (Teil 4): BI-Analysemethoden OLAP & Data Mining, [online] https://www.tecchannel.de/a/business-intelligence-teil-4-bi-analysemethoden-olap-und-data-mining,1739309 [Aufruf: 07.02.2021].

MongoDB (o. J. a): What is NoSQL? NoSQL Databases Explained | MongoDB, [online] https://www.mongodb.com/nosql-explained [Aufruf: 29.01.2021].

MongoDB (o. J. b): PyMongo – MongoDB Drivers, [online] https://docs.mongodb.com/drivers/pymongo [Aufruf: 28.10.2020].

MongoDB (o. J. c): db.collection.find() – MongoDB Manual, [online] https://docs.mongodb.com/manual/reference/method/db.collection.find/ [Aufruf: 01.02.2021].

MongoDB (o. J. d): Enable Access Control – MongoDB Manual, [online] https://docs.mongodb.com/manual/tutorial/enable-authentication/ [Aufruf: 29.10.2020].

MongoDB (o. J. e): Indexes – MongoDB Manual, [online] https://docs.mongodb.com/manual/indexes/ [Aufruf: 05.02.2021].

MySQL (o. J.): Chapter 1 Introduction to MySQL Connector/Python, [online] https://dev.mysql.com/doc/connector-python/en/connector-python-introduction.html [Aufruf: 05.02.2021].

Nguyen, Hoang (2018): Datenmodell: Sternschema, [online] https://data-science-blog.com/blog/2018/05/02/datenmodell-sternschema/ [Aufruf: 24.11.2020].

Paessler (o. J.): IT Explained: SNMP, [online] https://www.paessler.com/it-explained/snmp [Aufruf: 19.11.2020].

ProgramCreek (o. J.): Python scapy.all.IP Examples, [online] https://www.programcreek.com/python/example/103589/scapy.all.IP [Aufruf: 04.01.2021].

Rathish (2017): Difference between star schema and data cube?, [online] https://dba.stackexchange.com/a/172918 [Aufruf: 07.02.2021].

ReviverSoft (o. J.): .PCAP Dateierweiterung, [online] https://www.reviversoft.com/de/file-extensions/pcap [Aufruf: 07.02.2021].

Scapy (o. J. a): General Documentation | Introduction, [online] https://scapy.readthedocs.io/en/latest/introduction.html [Aufruf: 02.11.2020].

Scapy (o. J. b): General Documentation | Layers NetFlow, [online] https://scapy.readthedocs.io/en/latest/api/scapy.layers.netflow.html [Aufruf: 02.11.2020].

SolarWinds (2018): How to Configure NetFlow for Cisco Routers and Switches Running IOS, [online] https://www.youtube.com/watch?app=desktop&v=TZUW5lqzZDc [Aufruf: 24.08.2020].

SolarWinds (o. J.): Was ist NetFlow?, [online] https://www.solarwinds.com/de/netflow-traffic-analyzer/use-cases/what-is-netflow [Aufruf: 24.08.2020].

StackOverflow (o. J.): Replacements for switch statement in Python?, [online] https://stackoverflow.com/questions/60208/replacements-for-switch-statement-in-python [Aufruf: 16.12.2020].

Stetic (o. J.): Crontab Syntax und Tutorial, [online] https://www.stetic.com/developer/cronjob-linux-tutorial-und-crontab-syntax/ [Aufruf: 08.12.2020].

Wikimedia Foundation (2010): Simple Network Management Protocol, [online] https://deacademic.com/dic.nsf/dewiki/1291390#Party-Based_SNMP_Version_2_.28SNMPv2p.29 [Aufruf: 19.11.2020].

Winkler, Janina (2017): Was ist ein Portscan?, [online] https://www.biteno.com/was-ist-ein-portscan/ [Aufruf: 13.02.2021].

Tabellenverzeichnis

Abbildungsverzeichnis

Stichwortverzeichnis