HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ins institute for networked solutions

PeeringBox

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

FS 2013

Autoren:	André Ulrich / Reto Gsell
Betreuer:	Dipl. El. Ing. FH Rolf Schärer
Projektpartner:	SwissIX Glattbrugg
Experte:	Dipl. Inf. Ing. FH Michael Schneider
Gegenleser:	Prof. Dipl. El.-Ing. ETH Eduard Glatz

PeeringBox

Technischer Bericht
Bachelorarbeit

Autoren:	André Ulrich
	Reto Gsell
Betreuer:	Dipl. El. Ing. FH Rolf Schärer
Projektpartner / Auftraggeber:	Basile Bluntschli, SwissIX
Gegenleser:	Prof. Dipl. El.-Ing. ETH Eduard Glatz
Experte:	Dipl. Inf. Ing. FH Michael Schneider, wlan-partner.com AG
Abteilung:	Informatik
Themengebiet:	Internet-Technologien und -Anwendungen
Institut:	Institute for Networked Solutions (ins)
Zeitraum:	FS 2013 (18.02.2013 – 14.06.20013)

Änderungsgeschichte

Datum	Version	Änderung	Autor
01.06.2013	0.1	Merge aller Dokumente	André Ulrich
02.06.2013	0.2	Management Summary und Abstract verfasst	André Ulrich
02.06.2013	0.3	Korrektur lesen	Reto Gsell & André Ulrich
05.06.2013	0.4	Erweiterung Implementationsdokumentation	Reto Gsell & André Ulrich
07.06.2013	0.5	Tests & Metrik	Reto Gsell & André Ulrich
08.06.2013	0.6	Erfahrungsbereichte & Anhang	Reto Gsell & André Ulrich
10.06.2013	1.0	Erste Begutachtung durch Betreuer.	Reto Gsell & André Ulrich
11.06.2013	2.0	Änderungen eingepflegt. Finalisieren zur zweiten Begutachtung durch Betreuer	Reto Gsell & André Ulrich
12.06.2013	3.0	Änderungen eingepflegt. Finalisieren zur dritten Begutachtung durch Betreuer	Reto Gsell & André Ulrich
13.06.2013	4.0	Überarbeitung Kopf/Fuss-Zeilen, Finale Ausgabe für Druck	Reto Gsell & André Ulrich

Unterschriebene Aufgabenstellung

Ausschreibung:
PeeringBox

Internet Service Provider (ISP) partizipieren oft an sogenannten Internet Exchanges (IX). Dabei werden zwischen verschiedenen ISP's Peerings abgeschlossen, um kostengünstig und effizient Internetverkehr austauschen zu können. Eine Problematik die dabei entstehen kann ist, dass die ISP's nicht genau wissen, wieviel Verkehr mit welchem Peer ausgetauscht wird. Weiter ist es schwierig festzustellen, ob ein ISP an einem IX auch nur Verkehr von seinen Peering-Partner erhält oder ob er zusätzlich von anderen ISP's Verkehr statisch weitergeleitet erhält.

Verschiedene Internet Exchanges bieten dazu graphische Tools an, welche das eine oder andere Problem lösen können. Genaue Aussagen um weiterführende Verkehrsanlaysen vorzunehmen, sind aber meistens nicht möglich.

In diesem Projekt soll ein System designed und ein Produkt entwickelt werden, welches solche Problemstellungen modular und mandantenfähig abbilden kann. Dazu müssen Daten aus verschiedene Informationsquellen auf einem zentralen System gespeichert werden. Zu diesen Quellen zählen z.B.: Flowdaten von Netzwerkgeräten (sFlow), Interfaceinformationen von Netzwerkgeräten (via SNMP). Wir erwarten eine klar strukturierte, skalierbare Lösung welche mit einer grossen Menge an Daten umgehen kann. (100Gbps Verkehr resultiert in ca 100Mbps Sampledaten die ausgewertet, berechnet und gespeichert werden müssen)

Die vorausgehende Studienarbeit mit dem gleichen Titel, konnte einige Prototypen Fragen lösen, diese werden als Grundlage für die Weiterentwicklung dienen. Das Ziel der Arbeit ist ein lauffähiges Produkt zu erstellen, welches am SwissIX verwendet und den Kunden zur Verfügung gestellt werden kann.

Quelle: avt.hsr.ch

Rapperswil, 19.3.13	Rapperswil, 19.3.13
Ort, Datum	Ort, Datum
Rolf Schärer	Basile Bluntschli

Abbildung 1: Scan unterschriebene Aufgabenstellung

Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben. Ausgenommen sind externen Ressourcen mit Quellenangabe.

Ort, Datum	Ort, Datum
André Ulrich	Reto Gsell

Abstract

An einem Internet Exchange (IX) versammeln sich Organisationen, deren gemeinsames Interesse darin besteht, Daten direkt untereinander auszutauschen. Provider jeglicher Art können über die Infrastruktur eines IX Traffic direkt zum entsprechenden Ziel fliessen lassen und umgehen dabei anstehende Kosten möglicher Drittanbieter.

Im spezifischen Falle unseres Auftraggebers stellt die Non-Profit-Organisation „SwissIX“ eine Layer2-Netzwerkarchitektur zur Verfügung, welche aktuell 10 Standorte umfasst, die in der ganzen Schweiz verteilt sind.

Jeder einzelne Peer, der am SwissIX partizipiert, kann Verkehrsflussanalysen erstellen, was mit erheblichem Aufwand verbunden ist. Der IX selber kann hingegen relativ einfach qualitative Aussagen über den Verkehrsfluss zwischen Peers zur Verfügung stellen, welche von den verschiedenen Peers eingesehen werden können.

Im Rahmen der Bachelorarbeit wurde eine Java basierende Applikation mit Datenbank und Webapplikation als Frontend entwickelt, die genau diese Auswertungen präsentiert. Diese Applikation soll also sowohl den Angestellten des Internet Exchange (IX), sowie den verschiedenen Peers, welche untereinander Traffic austauschen, erlauben, Auswertungen über deren Datenfluss einzusehen.

Das System kann ohne grossen Pflegeaufwand arbeiten. Daten, die zur Verkehrsanalyse notwendig sind, werden automatisch von der Applikation zusammengetragen und in einer Datenbank konsistent gehalten. Somit reicht es, lediglich die verschiedenen Geräte (Layer2-Switch) im System, sowie Benutzerlogins zu erfassen. Alles Weitere läuft vollautomatisch ab. Sowohl der Initialzustand kann automatisch eruiert werden, wie auch Anpassungen im System werden detektiert und entsprechend protokolliert und umgesetzt.

Management Summary

Ausgangslage

Der SwissIX hat sich zum Ziel gesetzt, die Schweizer Provider besser untereinander zu vernetzen und dabei für alle Beteiligten Vorteile zu schaffen. Als Non-Profit-Organisation garantieren sie sogar, dass die Kosten lediglich den Aufwand decken sollen, der für den Unterhalt der Infrastruktur anfällt. Als grösster IX der Schweiz und aktuell 12 Jahren Bestehen (Gründung 9. März 2001), hat sich SwissIX gut und stabil positioniert.

Als Provider muss man eine Strategie festlegen, wie Traffic geleitet werden soll, um dem Endkunden möglichst gute Konnektivität zu gewährleisten. Diese Konnektivität hat auch ihren Preis. Um als Provider optimal von einer Mitgliedschaft am SwissIX zu profitieren, werden „Verträge“ zwischen verschieden Peers abgeschlossen. Diese beinhalten ein Abkommen, wie Traffic von einem Netzwerksegment ins andere geleitet werden. Somit entfallen Kosten, die anstehen würden, wenn der Traffic über Umwege ins Zielsegment geleitet würde.

Aus technischer und administrativer Sicht ist so ein Abkommen über die Infrastruktur des SwissIX nur schwer zu realisieren. Es ist aber ein Anliegen der verschiedenen Peers zu wissen, welchen Traffic und wie viel Traffic sie von anderen Peers erhalten oder an andere Peers weiter leiten. Provider können diese Auswertungen in eigener Sache erstellen, was viele auch machen. Dies ist jedoch mit Kosten verbunden und für kleine Unternehmen nur schwer zu realisieren.

Am Einfachsten und Effektivsten für alle Beteiligten ist es, wenn der IX direkt eine Plattform anbietet, in der alle Daten ersichtlich sind. Nicht nur, weil der Installationsaufwand dann lediglich an einer Stelle auftritt, sondern auch weil der direkte Zugriff auf die Netzwerkgeräte viel bessere und einfacher zu erstellende Aussagen ermöglicht.

Gewisse IX bieten dafür Anwendungen, doch diese sind auf dem freien Markt nicht erhältlich oder werden zu horrenden Preisen gehandelt.

Im Rahmen dieser Bachelorarbeit soll das Produkt PeeringBox diesen Missstand beseitigen und Transparenz für alle beteiligten Peers schaffen.

Vorgehen, Technologien

Da die verschiedenen Parteien, mit Ausnahme von Basile Bluntschli, bereits im vergangenen Semester die Studienarbeit in derselben Konstellation erfolgreich abgeschlossen hatten, waren Kontakte bereits geknüpft und das Themengebiet bereits vor Beginn der Bachelorarbeit klar. Einige Wochen vor Beginn wurden uns Informationen per Mail zugeschickt, damit wir uns in die Thematik eines IX einlesen konnten.

Zum Zeitpunkt der Studienarbeit wurde ein Redmine-Server bestellt, der es uns ermöglichte, stets über den aktuellen Stand des Projektes im Bild zu sein und unseren Aufwand auf die verschiedenen Teile des Projektes zu buchen. Dieser bestand also bereits und wurde für die Bachelorarbeit direkt übernommen, wodurch lediglich ein zusätzliches Projekt im System zu erfassen war.

Da unsere Bachelorarbeit an eine Studienarbeit, die quasi als Machbarkeitsstudie endete, anknüpft, wurde von uns bereits früh verlangt die Funktionsweise eines IX zu beschreiben, damit sichergestellt werden konnte, dass die Thematik verstanden wurde und somit keine Verständnisprobleme bei der Umsetzung vorhanden sind. Aus diesem Grund wurde gleich zu Beginn der Arbeit eine Funktionsanalyse erstellt, welche weiter unten im Punkt „Technische Beschreibung der Funktionsweise eines IX“ zu finden ist.

Des Weiteren hatten wir Einblick in die Dokumentation dieser Machbarkeitsstudie und konnten einige Erkenntnisse daraus übernehmen.

In einem nächsten Schritt mussten passende Produkte eruiert werden, die ein erfolgreiches, funktionsfähiges Produkt mit breiter Systemkompatibilität ermöglichen. Dies hat sich in folgenden Produkten manifestiert:

- Datenbanksystem: PostgreSQL
- Anwendungsserver: GlassFish
- sFlow-Collector: pmacct

Als Programmiersprache standen Java und JavaScript in Kombination mit html und css fest. Dabei stützen wir uns auf folgende Libraries und Technologien:

- Java Enterprise Beans (EJB)
- Java Server Pages (JSF)
- PrimeFaces
- JPA
- EclipseLink
- SNMP4J
- JavaMail
- NVD3, das auf d3.js aufbaut

Zuerst konzentrierte man sich in erster Linie auf das Backend. Dazu gehören Punkte wie Datenbankanbindung, sFlow-Datenkollektion, sowie Datenabfrage wie SNMP und Persistierung.

Sobald das Backend stand und mit den erwarteten Daten übereinstimmte, machten wir uns an die Implementation des Webinterfaces, was der Interaktion mit dem System und der Darstellung der verschiedenen Auswertungen dient.

Ergebnisse

Alle geforderten Funktionen, mit Ausnahme eines optionalen Punktes (Alarme), konnten zuverlässig implementiert werden. Die Daten stimmen mit den Annahmen und den Auswertungen des Observierungs- und Monitoring-Systems „Observium“ überein und entsprechen auch den Annahmen des Kunden.

Das System hat leider ein wenig mit den zur Verfügung stehenden Ressourcen zu kämpfen. Ohne Benutzerinteraktion läuft alles reibungslos ab. sFlow Daten können gesammelt, aggregiert und persistiert werden. Interface-Informationen können ebenfalls problemlos in knappem Intervall (5min) via SNMP abgefragt und persistiert werden. Bei Abfragen der gesammelten Daten über einen längeren Zeitraum jedoch müssen Millionen von Datensätzen durchforstet werden, wobei der Server an seine physikalischen Grenzen bezüglich Festplatten-I/O stösst. Dabei können je nach Abfrage Ladezeiten von mehreren Sekunden auftreten. Das Ziel war es, die Daten in feiner Granularität und entsprechendem Informationsgehalt zu speichern. Nach einer Analyse der Systemperformance wurden die Datenmengen auf Kosten des Informationsgehalts reduziert. Dadurch konnten die Ladezeiten auf der Webseite reduziert werden. Das System ist auf grosse Datenmengen angewiesen und somit auch auf entsprechende Hardware. Sobald das System auf besserer Hardware läuft, gehören diese Wartezeiten der Vergangenheit an.

Wie man sich das von einer Bachelorarbeit gewöhnt ist, überstieg auch hier wieder der Zeitaufwand den vorgesehenen Zeitrahmen um ein Vielfaches, doch dafür ist nun ein Produkt entstanden, das in eine erfolgversprechende Zukunft starten kann.

Ausblick

Im Rahmen der BA wurde das System lediglich für den Internet Exchange „SwissIX“ konzipiert. Dort arbeitet das System bereits mit Livedaten und analysiert bereits jetzt den Verkehrsfluss sämtlicher Teilnehmer.

In einer ersten Phase wird das System also lediglich den Mitarbeitern des SwissIX zur Verfügung stehen. Diese können damit den Teilnehmern qualitative Aussagen über die Aufteilung des Verkehrsflusses geben.

In einer zweiten Phase sollen die Peers eigene Logins auf der Seite erhalten. Damit können sie jederzeit Auswertungen über das eigene AS erstellen und dadurch Rückschlüsse auf ihre Partnerschaften mit anderen Peers ziehen.

In einer weiteren Phase ist es sogar angedacht das Produkt zu veröffentlichen. Dazu muss es jedoch um die Diversität von Netzwerkgeräten erweitert werden, denn aktuell können lediglich Informationen von Switches der Firma Brocade ausgewertet werden.

Inhaltsverzeichnis

1. Allgemein

1.1.1 Einführung

Dieses Dokument beinhaltet die gesamte Dokumentation der Bachelorarbeit „PeeringBox“. Die Bachelorarbeit „PeeringBox“ ist eine Webanwendung, welche es einem Kunden (ISP) ermöglicht den Verkehr, welchen er mit anderen Peers über einen IX austauscht, grafisch darzustellen. Jeder Mandant besitzt seine eigene Ansicht auf die Daten (spezifische Daten pro Mandant). Die Anwendung sammelt Daten von verschiedenen Switches, welche auf einem zentralen System gespeichert werden. Unter anderem werden sFlow- und SNMP-Daten von Netzwerkgeräten abgefragt. Die Anwendung bietet eine klar strukturierte und skalierbare Lösung, welche mit einer grossen Menge an Daten umgehen kann.

1.1.2 Gültigkeitsbereich

Das Dokument ist für die komplette Dauer der BA „PeeringBox“ gültig. Falls inhaltliche Änderungen am Dokument vorgenommen werden, muss dies dem Team mitgeteilt werden.

1.1.3 Fortsetzungsarbeit

Bei dieser Bachelorarbeit handelt es sich um eine Fortsetzungsarbeit einer SA. Die SA wurde mit demselben Ziel ausgeschrieben, hat jedoch in einer Machbarkeitsstudie geendet. Der Mehrwert aus der SA ist lediglich daraus entstanden, dass man sich schnell auf den sFlow-Collectors „pmacct“ geeinigt hat. Es wurde alles komplett neu erstellt und dabei keine einzelne Zeile Code übernommen.

Titel:	Internet Exchange Peering Box
Ersteller:	Wyss, Yannick
Typ:	Thesis (Student Research Project)
Datum:	15 Apr 2013
Veröffentlichung:	http://eprints.hsr.ch/247/

1.1.4 Einarbeitung

Die vorhergehende Arbeit wurde dem Projektteam bereits vor Beginn der Bachelorarbeit zur Verfügung gestellt. Zusätzlichen wurden die folgenden Links per Mail als Einarbeitungsmaterial zugesandt:

- https://www.euro-ix.net/
- https://www.youtube.com/watch?v=a5837LcDHfE
- http://de-cix.net/
- http://www.swissix.ch
- https://www.ams-ix.net/
- http://drpeering.net/
- http://www.nanog.org/presentations/archive/index.php

Diese Informationen wurden bereits vor dem 20. Februar

1.1.5 Referenzen

Alle im Dokument verwendeten Texte sind selbst geschrieben. Als Referenzen sind folgende Adressen zu nennen, die beim Aufbau des nötigen Knowhows behilflich waren:

- http://www.brocade.com
- http://www.primefaces.org/
- https://javamail.java.net/
- http://www.snmp4j.org/
- http://www.pmacct.net/
- http://nvd3.org/
- http://d3js.org/
- http://www.inmon.com/technology/index.php

2. Anforderungsanalyse

2.1 Einführung

Im Folgenden wird beschrieben, welche Anforderungen an das Produkt gestellt werden. Im frühen Stadium des Projektes musste erst verstanden werden, wie ein Internet Exchange (IX) funktioniert. Diese Erkenntnisse sind in diesem Abschnitt als Einstieg festgehalten um Lesern ein fundiertes Hintergrundwissen zu vermitteln.

Nachfolgend sind hier alle Anforderungen zu finden, welche vom Auftraggeber an die Software gestellt, bzw. von den Studenten bestimmt wurden.

Dieser Teil des Dokumentes wurde bereits in der Semesterwoche 5 vom Auftraggeber eingesehen und als komplett erachtet. Bei der Entwicklung des Produktes hat man sich also an die hier niedergeschriebenen Anforderungen gehalten.

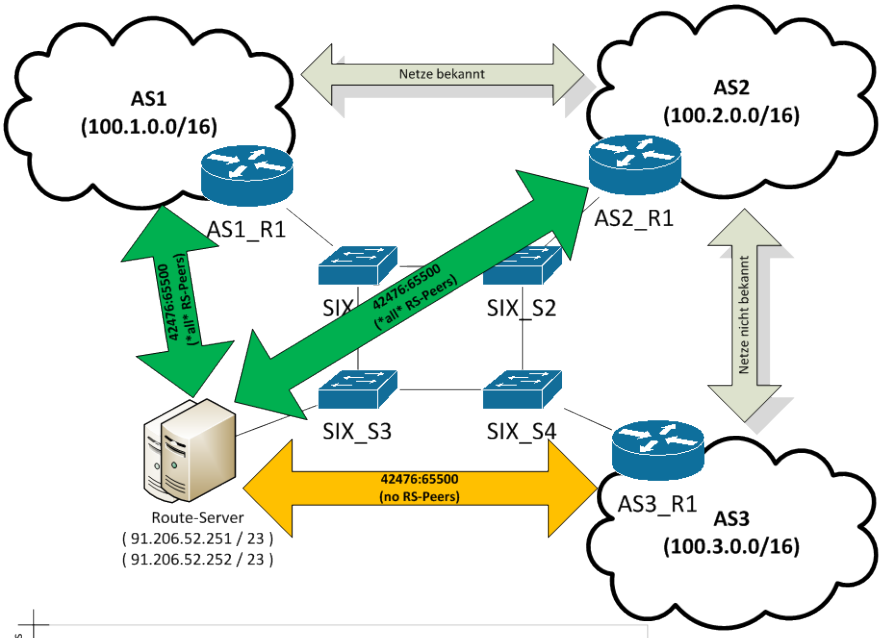
2.2 Technische Beschreibung der Funktionsweise eines IX

Der Internet Exchange „SwissIX“ ist hier lediglich schematisch als Verbund von 4 L2-Switches und einem Route-Server dargestellt. Des Weiteren sind die jeweiligen Peers lediglich mit einer fiktiven AS-Nummer und einem fiktiven IP-Range aufgeführt.

Die verschiedenen Bilder zeigen jeweils die entsprechende Thematik und beschreiben den Ablauf in textueller Form.

2.2.1 eBGP-Route-Server

Die erste Graphik beschreibt wie die Netze der einzelnen Peers (AS) untereinander bekannt gemacht werden, sofern sie dies wollen.



SwissIX - remarks

When peering with the SwissIX routeserver you can send communities to configure the certain redistribution behaviors:

42476:65500	Prepend RS AS42476 to *all* RS-Peers
6550n:neighbor-as	Prepend RS AS42476 to *selected* RS-Peer N-times (N = 1..3)
42476:neighbor-as	Do *not* announce to RS-Peers with AS neighbor-as
42476:65535	Do *not* announce to *ANY* RS-Peers

eBGP-Route-Server SetUp

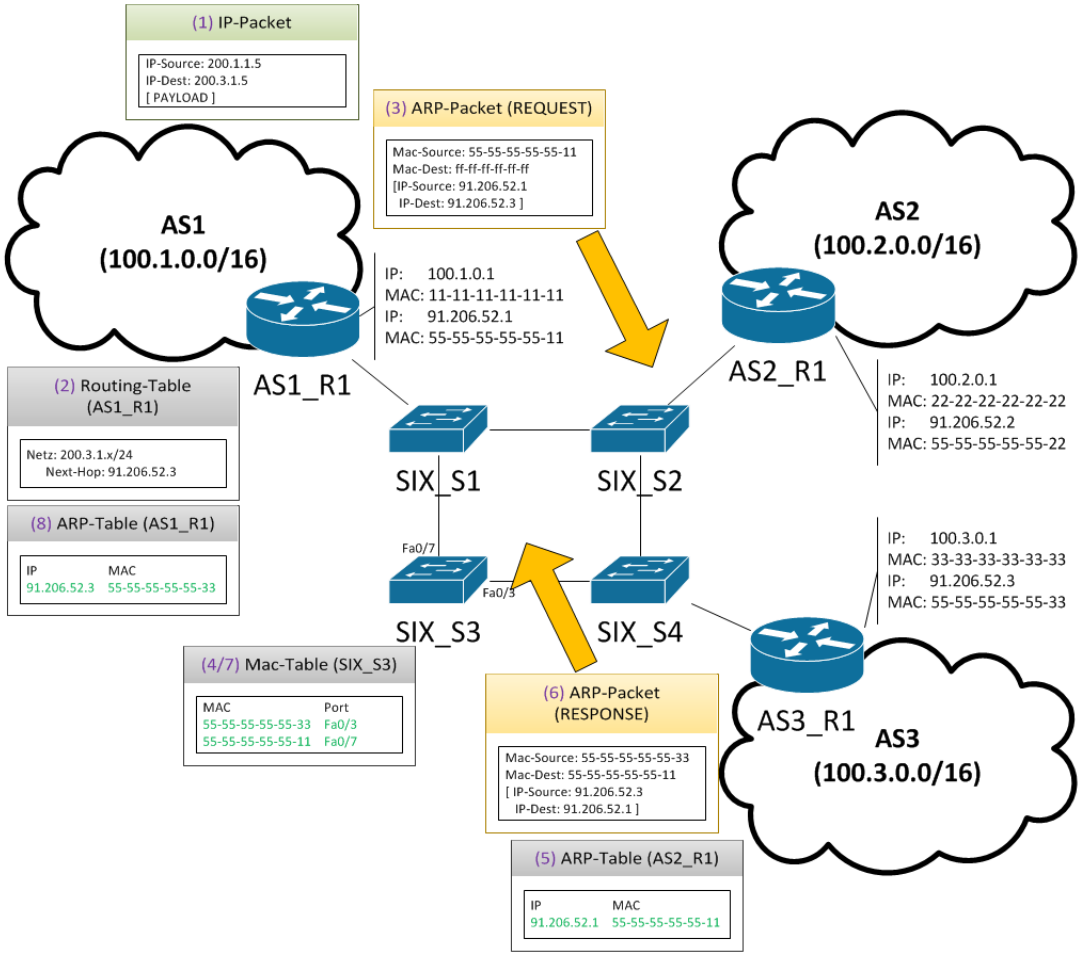
1. Der Route-Server spielt hierbei eine zentrale Rolle. Er «spricht» das Border Gateway Protocol (BGP) mit den verschiedenen Routern der verschiedenen autonomen Systeme (AS).
2. Die verschiedenen Router «announcen» ihr Netz dem Route-Server. Über den sogenannten Community-Namen können die einzelnen Teilnehmer bestimmen, welche Teilnehmer erfahren, dass das entsprechende Netz hinter diesem Router erreichbar ist.
Zum jetzigen Zeitpunkt kann immer noch keine Kommunikation stattfinden, da verschiedene AS (Farben) nichts voneinander wissen.
3. AS1 und AS2 haben sich für den Community-Namen entschieden, der es erlaubt, dass alle untereinander Pakete austauschen können.
4. Der Route-Server spricht sich nun mit AS1 und AS2 so lange ab, bis beide die notwendigen Informationen haben.
5. Ab diesem Zeitpunkt weiss AS1, dass er das Netz des AS2 via SwissIX erreichen kann.

Abbildung 2: eBGP-Route-Server

Natürlich können die verschiedenen Teilnehmer auch selber direkt untereinander peeren. Dies wird zumindest beim SwissIX erlaubt. Dabei wird lediglich die L2-Infrastruktur des Internet Exchanges genutzt. Alle weiteren Mechanismen müssen von den Teilnehmern selber konfiguriert werden.

2.2.2 Arp-/Mac-Table

Diese Graphik erläutert, wie die verschiedenen Router der Peers Mac-Adressen ihrer Kommunikationspartner erfahren. Natürlich lernen dabei auch die Switches von SwissIX die jeweiligen Adressen.



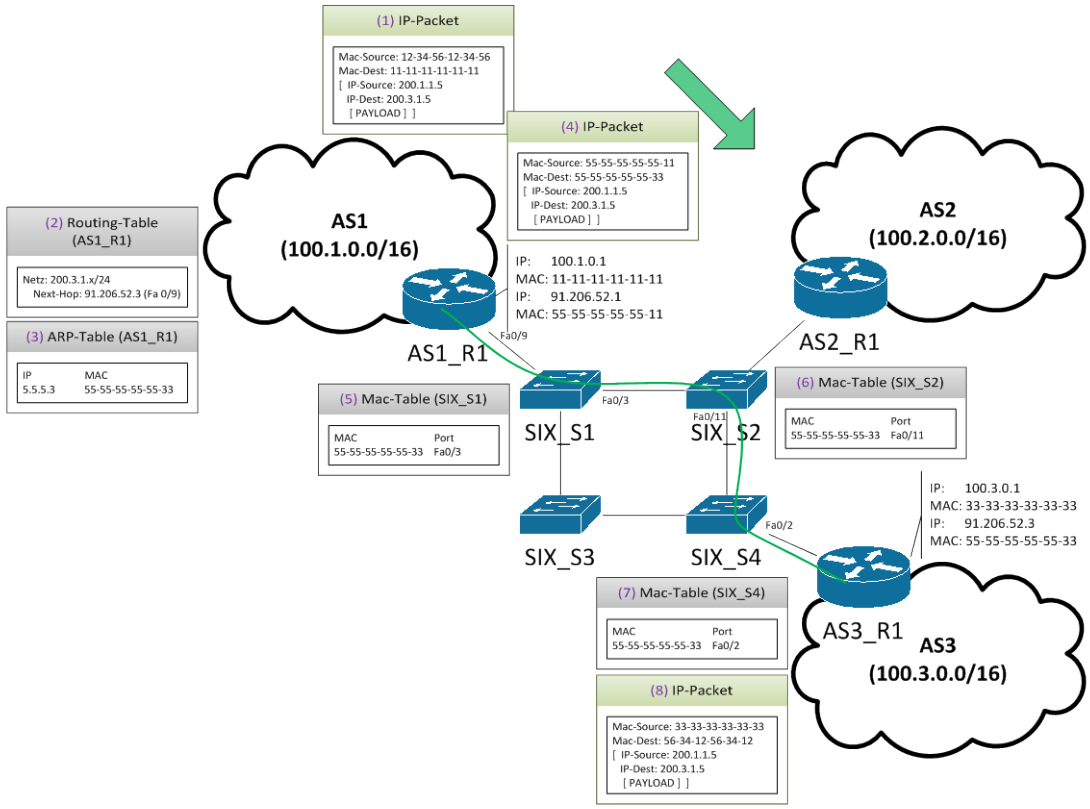
ARP- / Mac-Table

(1). Der Router des AS1 (AS1_R1) erhält ein IP-Paket. Als Ziel ist die Adresse 200.3.1.5 eingetragen.
(2). Der Router AS1_R1 schaut in seiner Routing-Table nach und erkennt, dass die Pakete in dieses Netz via Next-Hop (91.206.52.3) gesendet werden müssen.
(3). Dieser Hop ist dem Router noch nicht bekannt (kein Eintrag in der ARP-Table), weshalb er erst ein ARP-**Request** sendet. Dieses ist an die MAC ff-ff-ff-ff-ff-ff adressiert und wird an alle anderen Geräte gesendet.
(4). Sämtliche Geräte lernen nun, wo sich der Absender, jedoch immer noch nicht, wo sich der Empfänger befindet.
(5). AS3_R1 hat das Paket natürlich ebenfalls erhalten und trägt es in seiner ARP-Table ein.
(6). Zusätzlich sendet AS3_R1 eine ARP-**Response** an Router AS1_R1. Dieses Paket findet den Weg direkt.
(7). Alle durchlaufenen Switches lernen dabei die Adresse des Routers AS3_R1.
(8). AS1_R1 hat nun gelernt, welche Mac-Adresse zu der gesuchten IP-Adresse gehört und dies in die ARP-Table eingetragen.

Abbildung 3: Arp-/Mac-Table

2.2.3 Paketversand

Unten abgebildet ist nun der Weg eines Paketes, das zwischen dem AS1 und dem AS2 ausgetauscht wird. Dabei werden die eben gelernten Adressen zur Adressierung verwendet.



Paketversand

(1). AS1_R1 erhält ein Paket
(2). AS1_R1 schaut in seiner RoutingTable und erfährt, dass sein Next-Hop für das Ziel-Netz die IP 5.5.5.3 ist.
(3). Die entsprechende Mac-Adresse wird aus der ARP-Table entnommen.
(4). AS1_R1 schreibt die L2-Informationen des Paketes um. Das Paket wird dem SIX_S1 übergeben.
(5). In der Mac-Adress-Table auf SIX_S1 wird ersichtlich, dass sich das Gerät mit der Adresse 55-55-55-55-55-33 an Port Fa0/3 befindet. -> weiter leiten
(6). In der Mac-Adress-Table auf SIX_S2 wird ersichtlich, dass sich das Gerät mit der Adresse 55-55-55-55-55-33 an Port Fa0/11 befindet. -> weiter leiten
(7). In der Mac-Adress-Table auf SIX_S4 wird ersichtlich, dass sich der Zielrechner mit der Adresse 55-55-55-55-55-33 an Port Fa0/2 befindet. -> weiter leiten
(8). Paket kommt beim Router AS3_R1 an. Dieser schreibt wiederum die L2-Informationen um, sodass es innerhalb des AS3 seinen Zielort findet.

Abbildung 4: Paketversand

2.2.4 Problematik Verkehrsanalyse als Peer

Nun, da die Thematik erläutert ist, kann auch verstanden werden, warum es für einen Peer schwierig ist, qualitative Aussagen über den Verkehrsfluss zu machen.

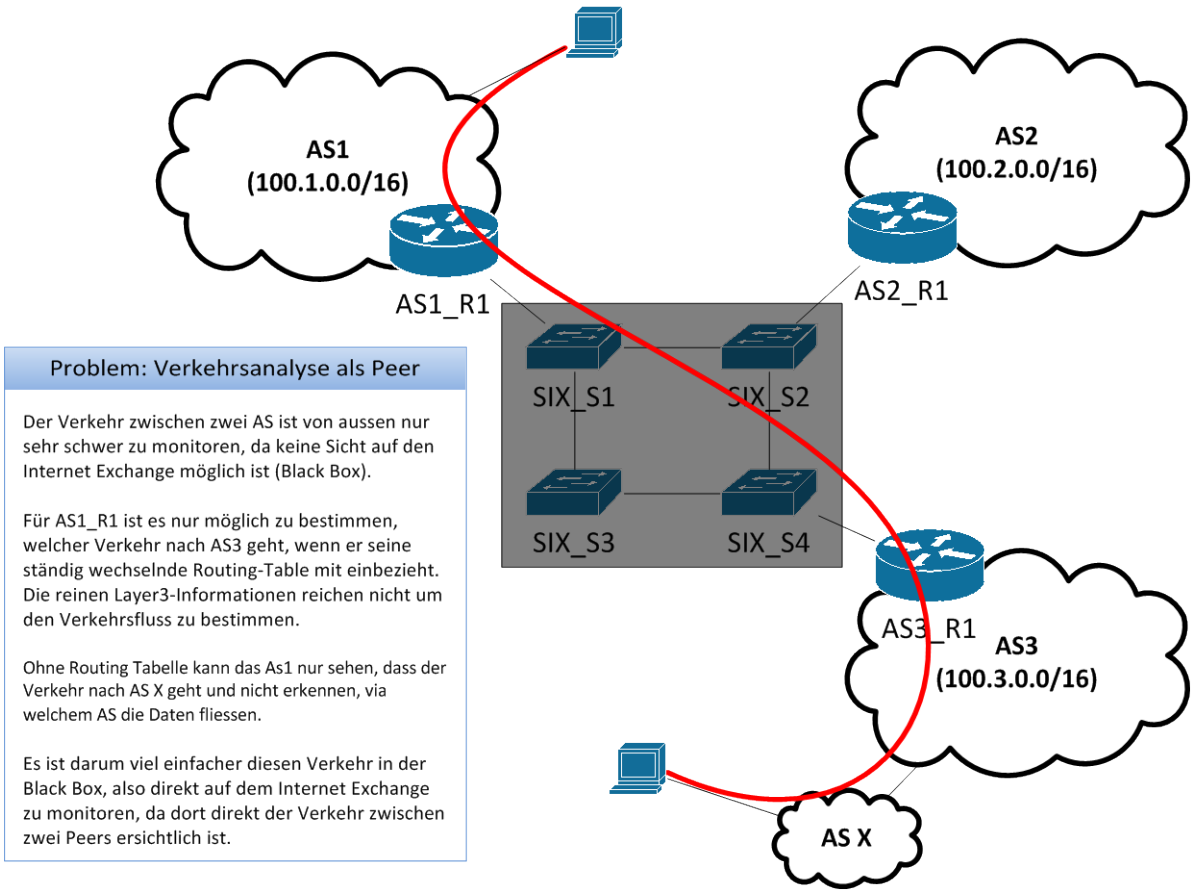


Abbildung 5: Problematik Verkehrsanalyse

Dies zeigt auf, welches Potenzial und welcher Mehrwert von einer solchen Applikation ausgehen.

2.3 Allgemeine Beschreibung

2.3.1 Produkt Perspektive

Verschiedene IX haben Tools im Einsatz, welche einen Teil der Problemstellung lösen. Genaue Aussagen um weiterführende Verkehrsanalysen vorzunehmen, sind aber meistens nicht möglich. Eigene Implementationen, die alle Anforderungen abdecken würden, sind nicht frei erhältlich. Die PeeringBox soll ein einfach zu bedienendes Tool sein, welches alle folgenden Anforderungen abdeckt.

2.3.2 Produkt Funktion

Webanwendung:

- Administrator kann Netzwerkgeräte bearbeiten (CRUD).
- Administrator kann Mandanteninformationen (Kunden / AS) bearbeiten (CRUD).
- Administrator kann Benutzer bearbeiten/zuweisen (CRUD).
- Benutzer kann Passwort ändern.
- Benutzer kann verschiedene grafische Auswertungen anzeigen lassen.

Server:

- Peers an konfigurierten Netzwerkgeräten werden automatisch erkannt und dem richtigen Mandanten zugewiesen. Diese Informationen werden periodisch via SNMP gesammelt und ausgewertet.
- sFlow-Daten werden gesammelt und ausgewertet.
- Daten werden dynamisch verwaltet (Skalierbarkeit)

2.3.2.1 sFlow (sampling)

Die Verkehrsdaten werden nicht erstellt, in dem jedes Paket analysiert wird. Dies würde einen enormen Datenbestand bedeuten und eine ungeheure Menge an Operationen. Durch Samples wird eine Annäherung an das effektive Resultat erreicht. Zu Beginn beträgt die Samplerate 512, was heisst, dass jedes 512. Paket zusätzlich via sFlow an den Analyse-Server gesandt wird. Natürlich werden dabei sensitive Daten, wie der Payload entfernt und lediglich relevante Daten wie Headerinformationen archiviert.

Dabei soll eine Genauigkeit mit einer Abweichung im tiefen einstelligen Prozentbereich erreicht werden. Bei der immensen Anzahl an Paketen, welche die Switches durchströmen, kann auch ein akzeptabler Mittelwert bestimmt werden, in dem lediglich ein Teil der Pakete angeschaut wird. Die gesammelten Werte müssen nun mit der Samplerate multipliziert werden, damit die Zahlen mit den Effektivwerten übereinstimmen.

2.3.2.2 Tracking von Veränderungen an der Infrastruktur

Gewisse Informationen werden auf den verschiedenen Switches direkt hinterlegt. Beispielsweise wird in der Interface-Beschreibung hinterlegt, welcher Kunde, bzw. welches AS an diesem Port angeschlossen ist. Das System muss mit diesen Änderungen umgehen können und nachvollziehen, dass der Traffic, der an einem Port festgestellt wurde, nach einer Änderung womöglich einem anderen AS zugeordnet werden muss.

Dazu werden periodisch Informationen von den verschiedenen Geräten geladen und mit der Datenbank verglichen. So wird mit der Zeit eine Historie geschaffen, womit sich zu jedem Zeitpunkt bestimmen lässt, welchem Kunden (AS) der Port zugewiesen werden muss.

2.3.2.3 Datenpräsentation

2.3.2.3.1 Einführung

In der Netzwerkwelt werden Daten und Graphen in der Regel in folgenden Zeitspannen betrachtet:

- Letzte Stunde
- Der letzte Tag
- Die letzte Woche
- Der letzte Monat
- Das letzte Jahr
- Vergangene Jahre

Es gilt also immer, Verkehr in Abhängigkeit des Auftretens-Zeitpunktes zu analysieren. Es stellt sich nun also die Frage, welche Granularität bei welcher Zeitspanne Sinn macht. Wo die feine Granularität bei der Betrachtung des letzten Tages sehr wichtig ist, kann bei Jahresdaten problemlos summarisiert werden.

2.3.2.3.2 Archivierung

Daten, welche bereits älter sind, müssen nicht mehr in der anfänglichen Granularität vorhanden sein. Dies wurde eben in der Einführung oben weiter erläutert. Somit wird es möglich, Datensätze zusammenzufassen. Dadurch gehen Informationen verloren, jedoch interessieren diese zu einem späteren Zeitpunkt nicht mehr. Nach einem Jahr ist es beispielsweise nur noch interessant, wie viele Daten im Januar gegenüber Juni geflossen sind, nicht aber ob Traffic vermehrt in den Morgen-, oder Abendstunden aufgetreten ist. Dies ist auch nötig, da sonst die gespeicherte Datenmenge regelrecht explodieren würde. Die Datenmenge darf nur durch zusätzlichen IX Verkehr zunehmen. Ausgenommen davon sind langfristige Daten / Informationen (z.B. alte Jahresgraphen).

Ein Flow wird beispielsweise durch die Kombination aus MAC-Source, MAC-Destination, Switch, Port, L2-Information, L3-Information definiert. Datenfelder, wie die Paketgrösse oder Paketanzahl, können nun pro Flow summarisiert werden.

2.3.2.4 Visualisierung

Die gesammelten Daten sollen visuell dargestellt werden. In der X-Achse befindet sich stets die Zeit. In der Y-Achse ist dies meist die Datenmenge, die geflossen ist.

In der Netzwerkwelt sind meist Graphiken, wie die Untenstehenden, zu finden:

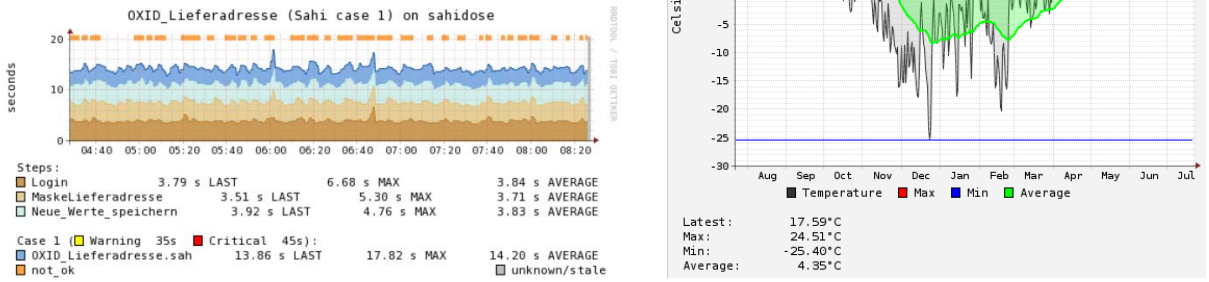


Abbildung 6: Beispiele von Graphen in der Netzwerkwelt[1]

Oft werden diese Bilder über eine Applikation wie RRDtool[2] erstellt.

Es ist jedoch nicht festgelegt, welche Applikation eingesetzt werden soll. Die Anforderung ist lediglich, dass die generierten Ansichten, den oben gezeigten Graphiken ähnlich sind. Viel Wert wird dabei darauf gelegt, dass mehrere Farben übereinander angeordnet werden können und sowohl der positive wie auch der negative Wertebereich genutzt werden können. Die Positive-/Negative-Darstellung soll hierbei für In-/Out-Verkehr auf demselben Graphen genutzt werden. Natürlich darf auch eine interaktive Lösung gewählt werden.

[1] Quelle: http://oss.oetiker.ch/rrdtool/gallery/index.en.html
[2] http://oss.oetiker.ch/rrdtool/

2.3.2.4.1 Views

Dies bezieht sich spezifisch auf die verschiedenen Graphiken, die generiert werden sollen. Es soll die Summe des gesamten Verkehrs angezeigt werden, getrennt mittels Flächen, unterschiedlichen Farben, nach Peers. Der Graph soll die eine Auswahl der Top-Peers inkl. Legende anzeigen. Die Farben für den In-/Out-Verkehr pro AS sind gleich (Sättigung kann angepasst werden). Bei multiplen Interfaces pro Peer, sollen diese an und abwählbar sein.

2.3.2.4.1.1 Default View

Beim Login wird eine Default View angezeigt. Die Default View zeigt einen Graphen der letzten Stunde.

- Drilldown auf einzelnen Peer, auf Summe aller restlichen Peers als 10ten Peer zusammenfassen.
- Darstellung Totaler Verkehr und Sub Total pro Peer.

2.3.2.4.1.2 Other Views

Other Views zeigt dasselbe wie die Default View. Anstatt der letzten Stunde stehen folgende Optionen zur Verfügung.

- Ansichten: (1h/24h/1week/4week/1year/(2year/3year/5year/10year))
- Durch den Kunden definierte Auswertung: per AS, per MAC oder per Name.
- Zeitspanne frei definierbar
- Subflächen für Protokoll Darstellung (Protocol per AS) <<optional>>

Gemäss den Berechtigungen im Benutzersystem (auf der Folgeseite definiert) sind gewisse Auswertungen den IX-Administratoren vorbehalten.

2.3.2.5 FrontEnd

2.3.2.5.1 Zugriff

Der Zugriff auf die Applikation soll über die Webbrowser Chrome, Safari und Firefox möglich sein. Es sollen aktuelle Webtechnologien eingesetzt werden, welche keine clientseitige Software benötigen (d.h. auch kein Flash oder Silverlight). Die Navigation soll einfach und intuitiv gestaltet sein. Der Verkehr zwischen Client und Server muss verschlüsselt sein (https).

2.3.2.5.2 Benutzersystem

Das System basiert auf Mandanten und Benutzern. Der Mandant stellt ein AS, Peer, bzw. Kunden des IX dar. Zwischen den Mandanten und Kunden besteht eine „N-zu-M“-Beziehung. Einem Mandanten können mehrere Benutzer zugeordnet und ein Benutzer kann zu mehreren Mandanten gehören. So können beispielsweise externe Berater die Verkehrsflüsse mehrerer Peers einsehen. Des Weiteren wird zwischen verschiedenen Benutzerrollen unterschieden. Zu Beginn sind normale Benutzer und Administratoren eingeplant.

Ein Angestellter eines Peers kann lediglich Daten einsehen, die den eigenen Peer betreffen, da in dieser Branche der Verkehrsfluss und die Datenmengen bestgehütete Geheimnisse sind. Mitarbeit des IX haben hingegen erweiterte Rechte und können sowohl Auswertungen für alle Peers wie auch Auswertungen über die gesamte Layer2-Architektur des Systems einsehen.

Jeder Benutzer hat folgende Rechte:

- Eigene Benutzerdaten bearbeiten
- Passwortwechsel
- Passwort zurücksetzen (Email/Web)
- Nur Zugriff auf eigene Daten (zugewiesenes AS)
- (optional) Alarm bei Abweichung vom täglichen/wöchentlichen Mittel vom jeweiligen Peer (email/Webseite)

Administratoren haben folgende zusätzlichen Rechte:

- Bearbeiten von Usern (CRUD)
- Einsicht in Auswertungen, welche den inneren Aufbau des IX betreffen. (Switch und Port-Auswertungen)
- Kann alle AS auswerten (inklusive Core Links)

2.3.2.5.3 Alarme (optional)

- Ein Alarm wird ausgelöst, wenn der Verkehr von einem AS, das kein Peering-Partner ist und mit mehr als 1Mbps Traffic auftritt. Alarm per Email und graphischer Anzeige des Vorfalls.
- Ein Alarm wird ausgelöst, wenn der Verkehr mit einem Peer nicht dem täglichen / wöchentlichen Mittel entspricht. Alarm per Email und graphische Anzeige des Vorfalls.

2.3.3 Benutzer Charakteristik

Alle Benutzer sind sowohl an aktuellen wie historischen Daten interessiert, die aussagen, mit welchen Unternehmungen, wie oft, wie viel und welche Art Traffic fliesst. Die Charaktere sind aber einfach in drei Gruppen zu unterteilen:

Peer-Mitarbeiter (User)
Hierbei handelt es sich um Mitarbeiter eines Unternehmens, das am SwissIX Traffic mit anderen Peers austauscht. Oft wird es pro Unternehmung einen Firmen-Account geben. Dieser Benutzer ist lediglich an Auswertungen interessiert, die sein Unternehmen (sein AS) betreffen. Möglicherweise kann ein Benutzer sogar berechtigt sein, die Auswertungen von mehreren AS einsehen zu dürfen. Dazu muss er den entsprechenden Unternehmen zugeteilt sein. Normale Benutzer sollen und dürfen aber lediglich die Daten einsehen, die ihre Unternehmung betreffen. Andere Ansichten sind gar nicht erst möglich.

Peer-Mitarbeiter mit erweiterten Rechten (Superuser)
Dieser Peer-Mitarbeiter hat dieselben Bedürfnisse wie ein normaler Peer-Mitarbeiter. Er hat aber das Privileg, User welche seinen eigenen Peers angehören zu verwalten.

IX-Mitarbeiter (Administrator)
Ein Mitarbeiter des Internet Exchange interessiert sich hauptsächlich für die Flüsse innerhalb des IX. Also welche Daten beispielsweise von Switch A nach Switch B fliessen. Um Kundenanfragen kompetent beantworten zu können, müssen diese Benutzer ebenfalls alle Daten einsehen können, die ein Kunde einsehen kann.

2.3.4 Einschränkungen

- Das Produkt arbeitet mit gesampelten Daten, welche von Netzwerkgeräten gesammelt werden. Darum wird keine 100-prozentige Genauigkeit erreicht werden können.
- Neue Netzwerkgeräte werden nicht automatisch erkannt und müssen somit erst erfasst werden. Dieser Schritt erfolgt bequem über das Userinterface, also per Web-Seite. (Weitere Informationen wie Portbeschreibungen, Mac-Adressinformationen, usw. werden automatisch gesammelt.)
- Die eingesetzte Software soll Open Source / Free to Use sein. Kostenpflichtige Software kann geprüft werden und mittels einer klaren Evaluation / Testing bei Basile angefragt werden.

2.3.5 Annahmen

- Zugriff via SNMP auf Netzwerkgeräte ist gewährleistet.
- Daten auf den Geräten werden vom IX Betreiber korrekt nachgeführt, um das korrekte Arbeiten der Anwendung zu gewährleisten.
- Das bereitgestellte Betriebssystem ist Debian.
- Der verwendete Webbrowser unterstützt sowohl HTML 4/5, CSS wie auch JavaScript.

2.3.6 Abhängigkeiten

Die Genauigkeit der Messungen hängt im weitesten Sinne von vier Faktoren ab:

- Die Sampelrate der sFlow Daten bestimmt, wie viele Messwerte ans System gesendet werden. Je höher diese ist, desto genauer können Auswertungen erfolgen.
- Das Traffic-Aufkommen bestimmt genauso wie die Sampelrate die Anzahl der Messwerte. Je mehr fliesst, desto aussagekräftiger werden die Messungen.
- Die Art der Daten ist ebenfalls ausschlaggebend. Wenn immer genau dieselben Daten durch den IX fliessen, ist eine Messung relativ einfach. Wenn jedoch vereinzelt riesige Datenpakete auftreten, dann kann es vorkommen, dass diese Pakete nicht erfasst werden und somit das Resultat der Auswertung verfälscht wird.
- Da sFlow-Daten via UDP versandt werden, kann es zu Verlusten von Daten kommen. Die Applikation ist also von einem funktionierenden Netzwerk abhängig.

Im Weiteren ist die Anwendung direkt von den Ressourcen des Servers abhängig. Dies macht sich hauptsächlich bei der Antwortzeit auf Auswertungen bemerkbar.

2.4 Use Cases

Folgend sind alle UseCases definiert und dokumentiert.

2.4.1 Use Case Diagramm

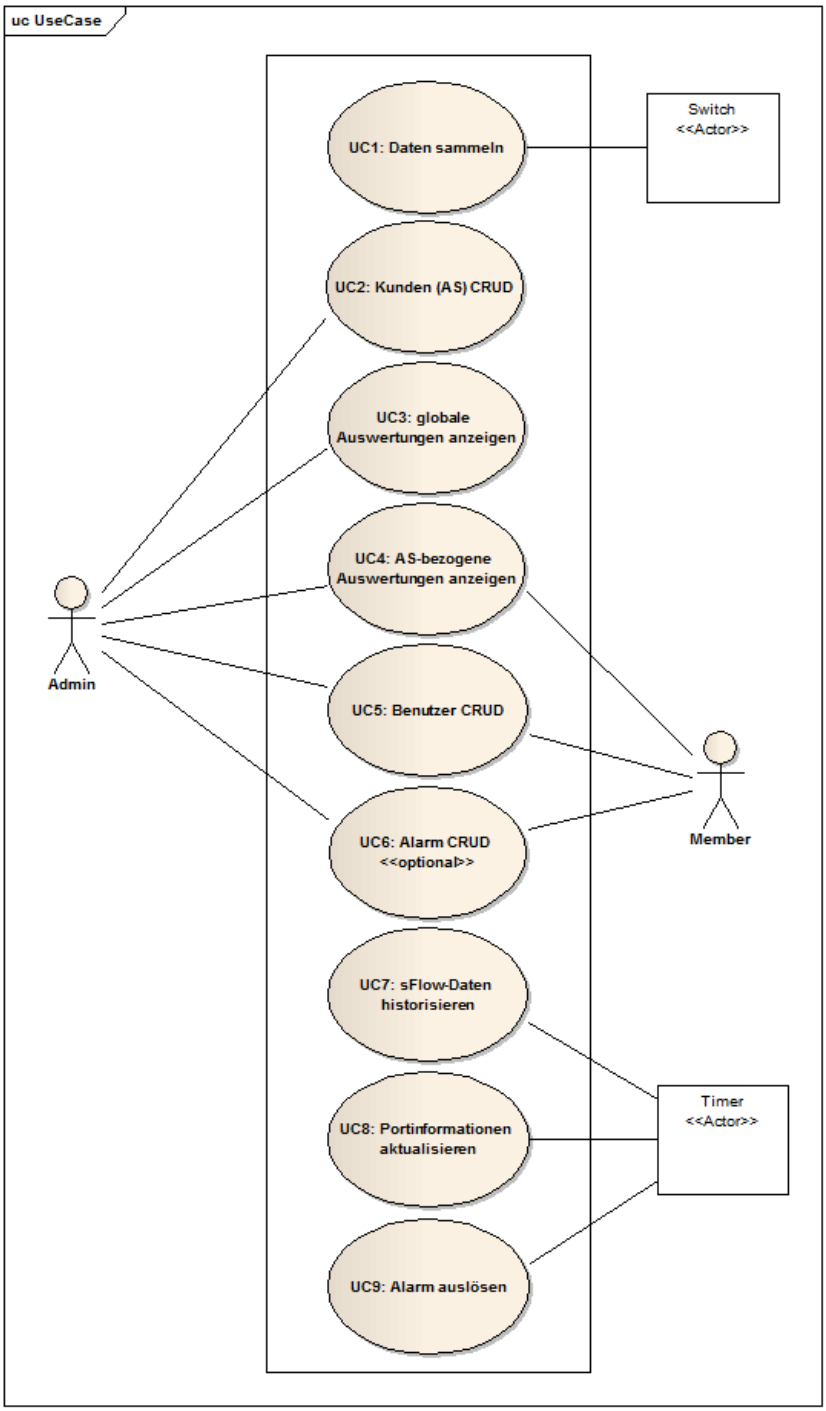


Abbildung 7: Use Case Diagramm

2.4.2 Aktoren & Stakeholder

- Admin Benutzer mit Administratorrechten (Mitarbeiter des IX/ SwissIX)
- Member normaler Benutzer (Mitarbeiter eines Kunden / Peering-Teilnehmer)
- Switch Die verschiedenen Switches, welche das Fundament des IX bilden.
- SwissIX-WebSeite Die Webseite, welche unter www.swissix.ch erreichbar ist.

2.4.3 Beschreibungen (fully dressed)

Im Folgenden sind die einzelnen Use-Cases, welche im oben eingebundenen Diagramm ersichtlich sind, genau beschrieben.

2.4.3.1 UC1: Daten sammeln

UC1: Daten sammeln	
Primary Actor	Die verschiedenen Switches dienen als Actor
Description	Der Switch übermittelt alle Sampels dem sFlow-Collector. Diese Daten bilden die Grundlage aller Auswertungen.
Stakeholder and Interests	• Admin • Member
Preconditions	1. Die Switches sind fertig konfiguriert. a. Die Samplerate wurde definiert. (z.B.: jedes 512. Paket) b. Ziel-IP und Port sind richtig konfiguriert. 2. Der SFlow-Collector auf dem Server ist so konfiguriert, dass Daten entgegengenommen werden können und automatisch in die Datenbank geschrieben werden.
Postconditions	• Sämtliche relevanten Daten wurden konsistent in der Datenbank abgelegt.
Main Success Scenario	1. Der Switch wartet auf Pakete. 2. Sobald der Switch den als Samplerate definierten Zähler erreicht hat, schickt er die Daten an die definierte Zieladresse. 3. Das Sample wird vom SFlow-Collector entgegen genommen. 4. Der Collector schreibt alle relevanten Daten, wie Mac-Adresse, Zeitstempel, usw. in die Datenbank.
Exceptions	2.a) Falls es nicht das gewünschte Paket ist, wird es nicht als Sample an den Server übermittelt.
Special Requirements	• Ein funktionierendes Netzwerk muss bereit stehen. Dieses muss IP und UDP unterstützen. • Der Rechner, der die Daten entgegen nimmt und die Datenbank zur Verfügung stellt, muss über genügend Performance verfügen, um mit der eintreffenden Datenflut umgehen zu können.
Technology and Data Variations List	Keine
Frequency of Occurrence	Dies hängt vom Verkehr innerhalb des Inter-Exchanges und der definierten Samplerate ab. Im Rahmen des SwissIX tritt ein Datenstrom zwischen 20 und 100 MBit/s auf. Dies entspricht mehreren hundert Samples pro Sekunde.

2.4.3.2 UC2: Kunden (AS) CRUD

UC2: Kunden (AS) CRUD	
Primary Actor	Admin
Description	Ein Kunde bezeichnet einen Peering-Teilnehmer. Alternativ ist von einem AS (autonomes System) die Rede. Einem AS sind diverse Benutzer (Member) zugeordnet. Diese müssen von den Administratoren verwaltet werden.
Stakeholder and Interests	• Member
Preconditions	• Der Administrator ist angemeldet und verifiziert. • Das AS ist bereits auf mindestens einem Switch in der Interfacebeschreibung hinterlegt.
Postconditions	• Die definierten Daten wurden im System gespeichert.
Main Success Scenario	1. Der Administrator navigiert in der Webapplikation zur Position, wo Kunden bearbeitet werden können. 2. Der entsprechende Kunde wird selektiert. 3. Folgende Felder können nach Belieben verändert werden: a. Member zuweisen b. Beschreibung erfassen c. Kontaktinformationen definieren / ändern 4. Der Administrator speichert die Eingaben durch einen Klick auf “speichern”.
Exceptions	Keine
Special Requirements	• keine
Technology and Data Variations List	keine
Frequency of Occurrence	Monatlich

2.4.3.3 UC3: globale Auswertungen anzeigen

UC3: globale Auswertungen anzeigen	
Primary Actor	Admin
Description	Administratoren sind neben den AS-bezogenen Auswertungen auch an globalen Auswertungen interessiert, welche den gesamten Internet-Exchange betreffen. Hier können beispielsweise Daten eingesehen werden, wie Verkehrsfluss zwischen Switch 3 und 4 oder Auslastung der einzelnen Switches.
Stakeholder and Interests	• Administratoren
Preconditions	• Der Administrator ist angemeldet und verifiziert. • Die gewünschten Daten sind in der Datenbank erfasst.
Postconditions	• Die gewünschte Auswertung wird angezeigt.
Main Success Scenario	1. Der Administrator navigiert in der Webapplikation zur Position, wo globale Auswertungen eingesehen werden können. 2. Er wählt, welche Daten er einsehen möchte. 3. Er wählt welchen Zeitraum er sehen möchte. 4. Die gewünschten Daten werden vom System aufbereitet und in Form von Text oder Bild angezeigt.
Exceptions	keine
Special Requirements	• keine.
Technology and Data Variations List	Keine
Frequency of Occurrence	Wöchentlich

2.4.3.4 UC4: AS-bezogene Auswertungen anzeigen

UC4: AS-bezogene Auswertungen anzeigen	
Primary Actor	Member
Description	Ein Kunde, bzw. ein Member, der einem AS (Kunden) zugewiesen ist, loggt sich ein, und betrachtet Auswertungen, welche sein AS betreffen.
Stakeholder and Interests	• Member
Preconditions	• Der Administrator ist angemeldet und verifiziert. • Die gewünschten Daten sind in der Datenbank erfasst
Postconditions	• Die gewünschte Auswertung wird angezeigt.
Main Success Scenario	1. Der Administrator navigiert in der Webapplikation zur Position, wo AS-spezifische Auswertungen eingesehen werden können. 2. Er wählt, welche Daten er einsehen möchte. 3. Er wählt welchen Zeitraum er sehen möchte. 4. Die gewünschten Daten werden vom System aufbereitet und in Form von Text, oder Bild angezeigt.
Exceptions	keine
Special Requirements	• keine
Technology and Data Variations List	Keien
Frequency of Occurrence	Stündlich

2.4.3.5 UC5: Benutzer CRUD

UC5: Benutzer CRUD	
Primary Actor	Admin
Description	Die Administratoren haben die Möglichkeit Benutzer zu verwalten (CRUD). Ein Benutzer kann mehreren Kunden (AS) zugeordnet werden. Der Benutzer kann einer Rolle zugeordnet werden (Administrator oder Member)
Stakeholder and Interests	• Admin • Member
Preconditions	• Administrator ist angemeldet und verifiziert.
Postconditions	• Ein neuer Benutzer wurde mit den richtigen Daten erstellt. • Zuweisungen an AS wurden korrekt erstellt.
Main Success Scenario	1. Der Administrator navigiert zur Benutzerverwaltung. 2. Der Administrator führt „neuen Benutzer erstellen“ aus. 3. Einstellungen für den Benutzer werden vorgenommen. 4. Der Benutzer wird mit seinen aktuellen Einstellungen gespeichert.
Alternative Flows	2a. Der Administrator wählt einen bestehenden Benutzer aus
Exceptions	keine
Special Requirements	• keine
Technology and Data Variations List	keine
Frequency of Occurrence	Zu Beginn sehr häufig, danach ca. 2-mal monatlich.

2.4.3.6 UC6: Alarm CRUD <<optional>>

UC6: Alarm CRUD <<optional>>	
Primary Actor	Member
Description	Ein Benutzer hat die Möglichkeit einen Alarm zu definieren. Dieser beinhaltet Schwellwerte und Aktionsmassnahmen. Bei Abweichungen vom normalen Flussverhalten soll das System den Benutzer informieren. Im Folgenden wird der Ablauf bei der Generierung eines neuen Alarms beschrieben.
Stakeholder and Interests	• Administrator • Member
Preconditions	• Benutzer hat sich im System angemeldet.
Postconditions	• Der Benutzer hat erfolgreich einen Alarm definiert.
Main Success Scenario	1. Der Benutzer navigiert in sein Profil. 2. Dem Benutzer werden alle bereits definierten Alarme präsentiert. 3. Der Benutzer klickt auf den Button „NEW ALARM“. 4. Es erscheint ein Dialog, der dem Benutzer die Möglichkeit gibt, die Schwellwerte der verschiedenen Parameter zu definieren. 5. Der Benutzer speichert den Alarm. 6. In der Übersicht von Punkt 2 wird der neue Alarm dargestellt.
Exceptions	keine
Special Requirements	keine
Technology and Data Variations List	keine
Frequency of Occurrence	Zu Beginn täglich, später eher selten.

2.4.3.7 UC7: sFlow-Daten historisieren

UC7: sFlow-Daten historisieren	
Primary Actor	Timer
Description	Der Timer leitet von Zeit zu Zeit einen Prozess ein, der Daten historisiert. Dabei summarisiert er Werte, speichert sie in die entsprechenden Tabellen und löscht die alten, verarbeiteten Werte. Somit sorgt er dafür, dass die Datenbank immer in etwa dieselbe Grösse behält. Es steht für jede Tabelle ein Prozess ablaufbereit. Dieser Prozess summarisiert die zusammengehörigen Daten der niedrigen Granularität in die nächst höhere Granularität.
Stakeholder and Interests	• Administrator • Member
Preconditions	• Daten müssen verfügbar sein.
Postconditions	• Die Daten wurden richtig historisiert.
Main Success Scenario	1. Die Uhrzeit erreicht den dafür vorgesehenen Zeitpunkt. 2. Der Timer stösst den entsprechenden Arbeits-Prozess an. 3. Das Ergebnis (Fehler oder Erfolg) wird in ein Log geschrieben. 4. Schritt 1, 2 und 3 werden stets wiederholt.
Exceptions	keine
Special Requirements	• keine
Technology and Data Variations List	keine
Frequency of Occurrence	Alle 10 Minuten

2.4.3.8 UC8: Portinformationen aktualisieren

3.3.8 UC9: Portinformationen aktualisieren	
Primary Actor	Timer
Description	Der Timer stösst automatisch von Zeit zu Zeit einen Prozess an, der sich via SNMP aktuelle Informationen von den Switches, bzw. deren Interfaces abholt. Dazu gehören beispielsweise das AS.
Stakeholder and Interests	• Administrator • Member
Preconditions	• Switches via SNMP erreichbar.
Postconditions	• Werte sind wieder aktuell.
Main Success Scenario	1. Die Uhrzeit erreicht den dafür vorgesehenen Zeitpunkt 2. Die Applikation arbeitet alle im System erfassten Switches ab und liest die Werte per SNMP aus. 3. Die Werte werden, sofern sie verändert wurden, in der Datenbank aktualisiert. 4. Das Ergebnis (Fehler oder Erfolg) wird in ein Log geschrieben. 5. Schritt 1 bis 4 wird stets wiederholt.
Exceptions	keine
Special Requirements	• keine
Technology and Data Variations List	keine
Frequency of Occurrence	Stündlich

2.4.3.9 UC9: Alarm auslösen <<optional>>

UC9: Alarm auslösen <<optional>>	
Primary Actor	Timer
Description	Der Timer arbeitet von Zeit zu Zeit alle Alarme ab. Sobald ein Schwellwert eines definierten Alarms überschritten wird, wird der entsprechende Benutzer informiert.
Stakeholder and Interests	• Administrator • Member
Preconditions	• Benutzer hat gewünschten Alarm definiert. • Benutzer hat eine E-Mail-Adresse hinterlegt • Der Server kennt alle Parameter, um E-Mails versenden zu können.
Postconditions	• Der Benutzer wurde über eine Abweichung informiert.
Main Success Scenario	1. Der Timer stösst den Prozess an, der alle Alarme prüft. 2. Der Prozess beginnt, die Liste aller Alarme sequenziell abzuarbeiten. 3. Der Prozess vergleicht die Werte mit dem im Alarm definierten Schwellwert. 4. Der Schwellwert wurde überschritten. Der Benutzer wird per E-Mail informiert. Zusätzlich bekommt er eine Notiz auf der Webseite. 5. Punkt 3 und 4 wiederholen sich, bis alle Alarme abgearbeitet sind.
Exceptions	4.a) Falls kein Schwellwert überschritten sein sollte, wird der Alarm nicht ausgelöst.
Special Requirements	keine
Technology and Data Variations List	keine
Frequency of Occurrence	Wöchentlich

2.5 Weitere Anforderungen

2.5.1 Mengenanforderungen

Aktuell liefern die Switches einen Datenstrom von ca. 100 Mbps. Diesen Datenstrom gilt es zu bewältigen. Zukünftig muss aber auch eine Datenmenge von 1000 Mbps verwaltet werden können. Es soll vorgesehen werden, dass mittels der gespeicherten sFlow Daten einfach weitere Auswertungen erstellt werden können.

2.5.2 Qualitätsmerkmale

Die folgenden Merkmale stellen eine Qualität gemäss ISO/IEC 9126 sicher.

2.5.2.1 Funktionalität

2.5.2.1.1 Angemessenheit

Um den Speicherbedarf der Anwendung möglichst tief zu halten, werden die Daten für die Historisierung zusammengefasst und nur noch nötige Daten gespeichert. Der Speicherbedarf wächst nur mit zunehmender Netzwerklast oder um die jährlichen Historien.

2.5.2.1.2 Richtigkeit/Genauigkeit

Durch das Sampeln der Datenströme gehen Informationen verloren, was sich auf die Genauigkeit der Messdaten auswirkt. Die Abweichung vom effektiven Durchsatz soll im tiefen einstelligen Prozentbereich liegen (Verglichen mit Standard SNMP ifcounter Abfragen - Observium).

Die Werte, welche in der Datenbank liegen, werden korrekt in der Visualisierung dargestellt. Ebenfalls werden keine Daten vertauscht oder falsche Daten einem Kunden angezeigt.

2.5.2.1.3 Interoperabilität

Durch eine serverzentrierte Lösung ist die Interoperabilität innerhalb des Systems gewährleistet. Da eine Webanwendung vorausgesetzt ist, muss die Funktionsfähigkeit mit den gängigen Browsern (Chrome, Safari und Firefox) gewährleistet sein.

Zudem soll die Möglichkeit in Betracht gezogen werden, verschiedene Komponenten auf unterschiedlichen Systemen zu betreiben (zum Beispiel: Collector 1 auf System 1, Collector 2 auf System 2, DB als Cluster auf verschiedenen Systemen, Auswertung auf System 3, Frontend auf System 4).

2.5.2.1.4 Sicherheit

Durch die Verwendung der PeeringBox sollen keine Sicherheitsdefizite entstehen. Die Webanwendung ist mit https gesichert. Dazu wird in erster Linie mit einem selbst zertifizierten Zertifikat gearbeitet. Dieses wird zu einem späteren Zeitpunkt durch ein offizielles Zertifikat ersetzt, liegt jedoch nicht im Rahmen der Bachelorarbeit. Passwörter werden stets nur in verschlüsselter Form (hash) abgelegt

2.5.2.2 Zuverlässigkeit

2.5.2.2.1 Reife

Das System erreicht eine geringe Versagenshäufigkeit durch Fehlerzustände. Trotz fehlerhafter Zustände im System wird die Applikation nicht versagen. Bei korrekt eingerichteten und funktionierenden sFlow-Geräten und laufendem sFlow-Collector sollen alle Daten korrekt in die Datenbank gespeichert und interpretiert werden.

2.5.2.2.2 Fehlertoleranz

Die Anwendung muss auch bei fehlenden Datenbeständen oder Operationen funktionsfähig sein und bleiben. Fehler können auftreten und werden ausgegeben, jedoch soll dies das System nicht in die Knie zwingen.

Beispiel: Fehlende Daten können bei einem Neustart oder Ausfall des Servers entstehen. In dieser Zeit können keine Daten gesammelt werden und doch soll eine Auswertung dieser Zeitspanne möglich sein. Die fehlenden Daten verursachen keinen Absturz.

2.5.2.2.3 Wiederherstellbarkeit

Falls das System durch Fehler oder äussere Umstände nicht mehr funktionsfähig sein sollte, soll der Betrieb einfach wiederhergestellt werden können. Notwendige Funktionalitäten werden automatisch neu gestartet, wodurch der Betrieb automatisch wieder aufgenommen werden kann.

2.5.2.3 Benutzbarkeit

2.5.2.3.1 Verständlichkeit

Das GUI der PeeringBox muss intuitiv zu bedienen sein.

2.5.2.3.2 Erlernbarkeit

Die Graphen sollen sich an den im Netzwerk-Umfeld üblichen Graphen (zum Beispiel: RRDTool) orientieren. Es muss einfach möglich sein, individuelle Graphen zu erstellen.

2.5.2.3.3 Attraktivität

Das visuelle auftreten der Clientsoftware muss möglichst ansprechend und schlicht gehalten werden. Das einheitliche und durchdachte Design muss dem Benutzer einen professionellen Eindruck hinterlassen.

2.5.2.4 Effizienz

2.5.2.4.1 Zeitverhalten

2.5.2.4.1.1 Server

Das Zeitverhalten der Serversoftware ist abhängig von der Konfiguration der Tools (Datenbank, Collector) und der Hardwarekonfiguration. Der Server muss in der Lage sein, mit grossen Datenmengen umzugehen.

2.5.2.4.1.2 Client

Das Zeitverhalten des Clients ist hauptsächlich von der Geschwindigkeit der Internetverbindung zum Webserver abhängig. Clientseitig werden keine komplizierten Berechnungen bei der Abfrage der Graphen durchgeführt. Lediglich die Berechnung der Graphik selbst findet auf dem Client statt, doch dies sollte mit handelsüblichen Rechnern keinerlei Probleme darstellen. Die Abfrage von Graphen sollte in einer angemessenen Zeit durchgeführt werden können.

2.5.2.5 Wartbarkeit

2.5.2.5.1 Analysierbarkeit

Durch Fehlerlogging auf dem Server können Ursachen von Versagen schnell diagnostiziert werden.

2.5.2.5.2 Modifizierbarkeit

Bei der Implementation wird Wert darauf gelegt, dass Änderungen und Erweiterungen in der Funktionalität schnell und einfach möglich sind. Dafür sorgt ein sauberes und durchdachtes Softwaredesign. Änderungen im Kernsystem sind jedoch mit mehr Aufwand verbunden, da alle restlichen Komponenten, die auf dieser Funktionalität aufbauen, entsprechend angepasst werden müssen.

2.5.2.5.3 Stabilität

Änderungen an der Datensammlung via sFlow und SNMP sollen keine unerwarteten Fehler produzieren. Nach einem Komplettausfall (z.B.: Stromausfall) wird die Applikation automatisch wieder in Betrieb genommen.

2.5.2.5.4 Testbarkeit

Die serverseitige Implementation der Software, bzw. deren Algorithmen und Funktionen können gut getestet werden. Dazu können automatisierte Tests implementiert werden, welche die Funktionsfähigkeit der Komponenten sicherstellt.

Tests der Webapplikation und der Datenbank können nicht ohne enormen Aufwand implementiert werden. Deshalb wird hier mit menschlicher Komponente getestet. Funktionen werden also vom Tester ausgelöst und die Reaktion des Systems von Hand nachvollzogen.

2.5.2.6 Übertragbarkeit

2.5.2.6.1 Anpassbarkeit

Die Serversoftware besteht aus einem sFlow Collector, einer Datenbank und einer Java Implementation. Abgesehen vom sFlowCollector, der auf ein Linux-System ausgelegt ist, wäre die Software auf verschiedenen Systemen lauffähig. Ein gesplitteter Betrieb wäre ebenfalls möglich. Vorausgesetzt ist aber eine Lauffähigkeit auf einem Linux (Debian) System.

2.5.2.6.2 Installierbarkeit

Die Serverinstallation besteht aus Java Applikation inklusive Libraries, Webserver, Datenbank und dem sFlow-Collector. Die Software soll ohne zu grossen Aufwand zu installieren sein. Zur Unterstützung dienen eine Installationsanleitung, sowie die Dokumentation.

2.5.2.6.3 Koexistenz

Für die Implementation wird ein individueller Server aufgesetzt. Demzufolge wird dafür keine entsprechende Anforderung erfasst.

2.5.3 Schnittstellen

2.5.3.1 Benutzerschnittstelle

Die Benutzerschnittstelle wird hauptsächlich durch die Webanwendung abgedeckt. Alle für den Kunden relevanten Funktionen sind komplett über den Browser bedienbar.

Die Konfiguration des sFlow Collectors muss über die entsprechenden Config-Files bedient werden. Hier wird auf die Dokumentation des Produktes „pmacct“ verwiesen.

2.5.3.2 Hardwareschnittstelle

Da die Anforderungen an die Hardware stark von der auftretenden Menge an Daten abhängen, muss die Skalierung gewährleistet sein. Falls die Leistung des normal konfigurierten Systems nicht genügt, muss das System der Software oder die Hardware angepasst werden. Dies bedeutet, dass physikalisch getrennte, schnelle Laufwerke oder RAM-Disks eingerichtet werden müssen. Die Hardwareschnittstelle wird durch das Linux Betriebssystem abgebildet.

2.5.3.3 Softwareschnittstellen

Der Server benötigt eine Anbindung an eine Datenbank, um die Messdaten zu verwalten. Diese Datenbank wird mit PostgreSQL umgesetzt. Zusätzlich ist dies sowohl pmacct und GlassFish.

2.5.4 Randbedingungen

2.5.4.1 Erweiterbarkeit

Es wird vorgesehen, dass mittels der gespeicherten sFlow-Daten einfach weitere Auswertungen erstellt werden können. Aus diesem Grund wird bei der Entwicklung viel Wert darauf gelegt, dass ohne grossen Mehraufwand weitere Funktionalitäten hinzugefügt werden können.

2.5.4.2 Serversoftware

Die Serversoftware wird mit Java 1.7 realisiert.

2.5.4.3 Zugriff auf Netzwerkgeräte

Der Zugriff auf die Netzwerkgeräte soll via SNMP gelöst werden. Der Zugriff über die CLI (Telnet/SSL) soll vermieden werden.

2.5.4.4 Backup

Es muss möglich sein, Backups des Systems zu erstellen. Dies ist durch die Datenbank relativ einfach zu realisieren. Es kann zu jeder Zeit ein SQL-Dump erstellt werden, welcher zu einem späteren Zeitpunkt wieder eingespielt werden kann.

2.5.4.5 Mutationen

Mutationen von Kunden, Ports, AS Nummern und MAC Adressen müssen gewährleistet werden. Die Auswertungen müssen diese Änderungen berücksichtigen und entsprechend bei der Abfrage mit einberechnen, damit eine Auswertung aller zum Peer gehörigen Daten möglich wird.

3. Domainanalyse

3.1 Übersicht

Dieser Abschnitt beschreibt die Analyse der Domain für das Projekt PeeringBox. In der objektorientierten Programmierung ist es von essentieller Wichtigkeit einen klaren Plan zu erstellen, um den Code logisch aufbauen zu können. Dies hilft nicht nur den Studenten bei der Implementierung, sondern auch möglichen Programmierern, welche die Software verwenden oder erweitern möchten.

3.2 Domain Modell (EJB)

Das Domain Modell beschreibt lediglich die sogenannten Real-World-Objects. Bezogen auf das Projekt PeerinBox sind dies die folgenden Objekte.

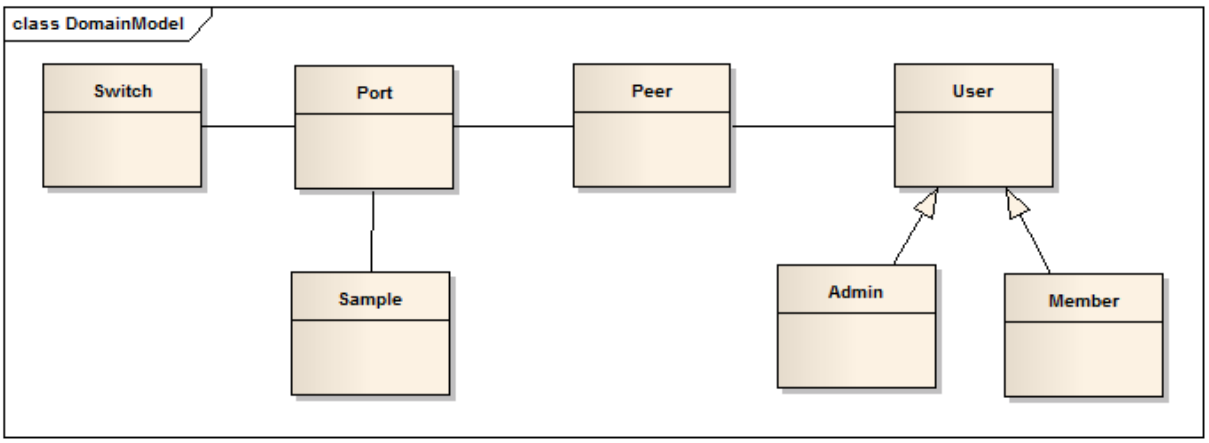


Abbildung 8: Domain Model

Eine kurze Erläuterung der einzelnen Objekte:

- Device — Physikalisches L2-Netzwerkgerät (Switch) des IX
- Port — Ein Port am Switch
- Sample — Gesammelte sFlow-Daten
- Peer — Ein Teilnehmer des IX (Internet Provider)
- User — Ein Login, das ermöglicht auf der Seite in den internen Bereich zu gelangen
- Admin — Ein User mit erhöhten Rechten
- Member — Ein User mit normalen Rechten, der in der Regel auch nur einem Peer zugewiesen ist.

3.3 Klassendiagramm

Das Klassendiagramm wird auf Grund des Domainmodels erstellt, ist aber bereits sehr viel konkreterer ausgearbeitet. Bei der Entwicklung der Anwendung hält man sich an dieses Design. Dies bezieht sich lediglich auf die Businesslogik, was bedeutet, dass bei der Implementation jede Menge weiterer Klassen erstellt werden.

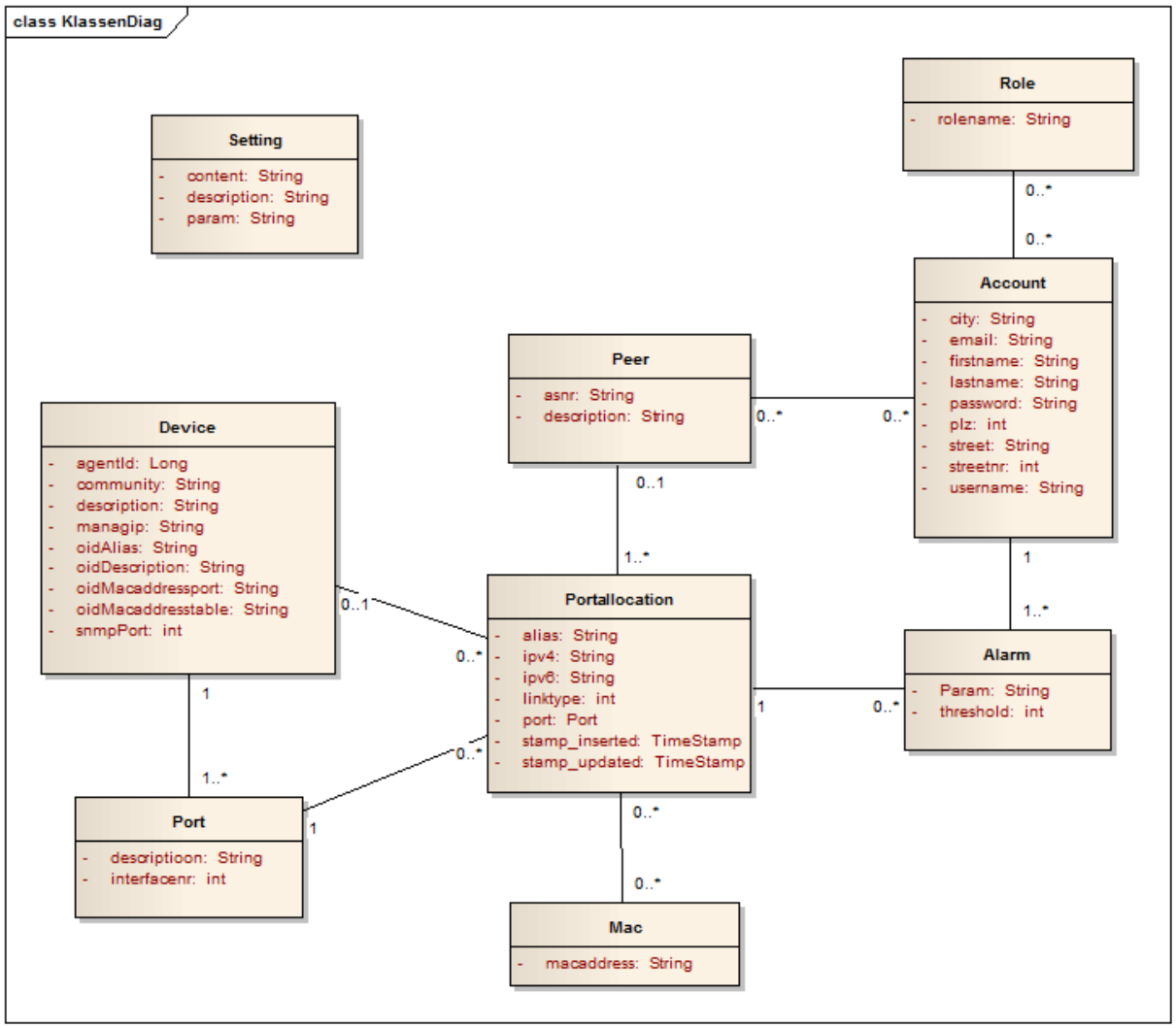


Abbildung 9: Klassendiagramm

3.3.1 Klassen (Entities)

Da die gesamten Laufzeit-Daten der Applikation persistent gehalten werden müssen, sind die verschiedenen Klassen ebenfalls Entities. Diese werden in der hier beschriebenen Form in der Datenbank gespeichert. Attribute und Klassen, die sich auf Grund des Namens erklären lassen, werden im Dokument nicht weiter erläutert.

3.3.1.1 Device

Ein Device bildet den physikalischen Switch des IX ab. Die Attribute managip, snmpPort, community, sowie die verschiedenen OIDs werden direkt verwendet, um mit dem Switch in Kontakt zu treten (SNMP) und benötigte Informationen abzurufen.

Die agentId wird verwendet, um das Gerät den verschiedenen Samples zuzuordnen. Diese ID muss dem post_tag in der pmacct Konfiguration des jeweiligen Gerätes entsprechen.

3.3.1.2 Portallocation

Eine Portallocation stellt ein Objekt dar, das alle Komponenten im System in einer bestimmten Zeitspanne miteinander verknüpft. Ein Timer, der bei Default-Konfiguration alle 5 Minuten abläuft, holt sich den gesamten Zustand des Systems und vergleicht ihn mit den Daten in der Datenbank. Sofern sich am Zustand der entsprechenden Portallocation nichts geändert hat, wird lediglich der Zeitstempel erhöht, womit markiert wird, dass dieser Eintrag noch aktuell ist. Bei Änderungen im System wird eine neue Portallocation erstellt. Die alte Portallocation bleibt im System erhalten und wird bei Auswertungen über vergangene Zeitspannen mit in die Berechnung einbezogen.

3.3.1.3 Role

Es gibt aktuell nur 3 Rollen im gesamten System. Diese sind Admin, User und Superuser. Diese Rollen legen fest, ob der Benutzer erweiterte Rechte hat und dadurch beispielsweise Devices verändern, hinzufügen oder bearbeiten kann.

Dies hat jedoch nichts mit der Berechtigung auf die verschiedenen Peers zu tun. Diese Berechtigung wird durch die Beziehung zwischen Account und Peer abgebildet.

3.3.1.4 Setting

Settings sind globale Systemvariablen. Pro konfigurierbarem Wert in der Applikation gibt es hier einen Eintrag. Diese Einstellungen werden im Userinterface abgebildet und werden nur Administratoren dargestellt. Diese Tabelle ersetzt mühsame Konfigurationsfiles und macht Einstellungen komfortabler erreichbar.

3.4 Systemsequenzdiagram

Folgender Abschnitt beschreibt die wichtigsten Abläufe der PeeringBox als Systemsequenzdiagram.

3.4.1 Peerauswertung erstellen

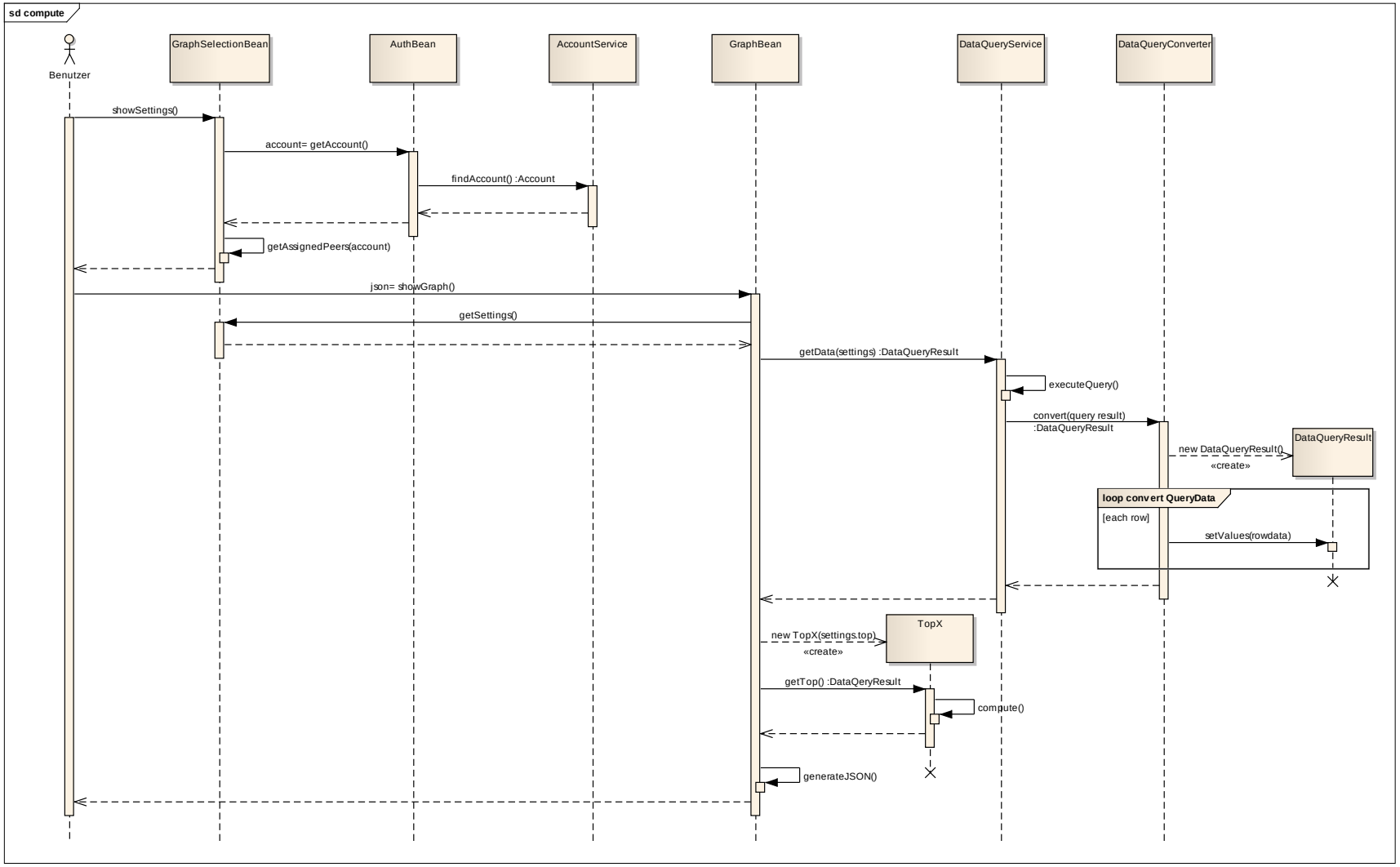


Abbildung 10: Sequenzdiagram Peerauswertung erstellen

3.4.2 Coreauswertung erstellen

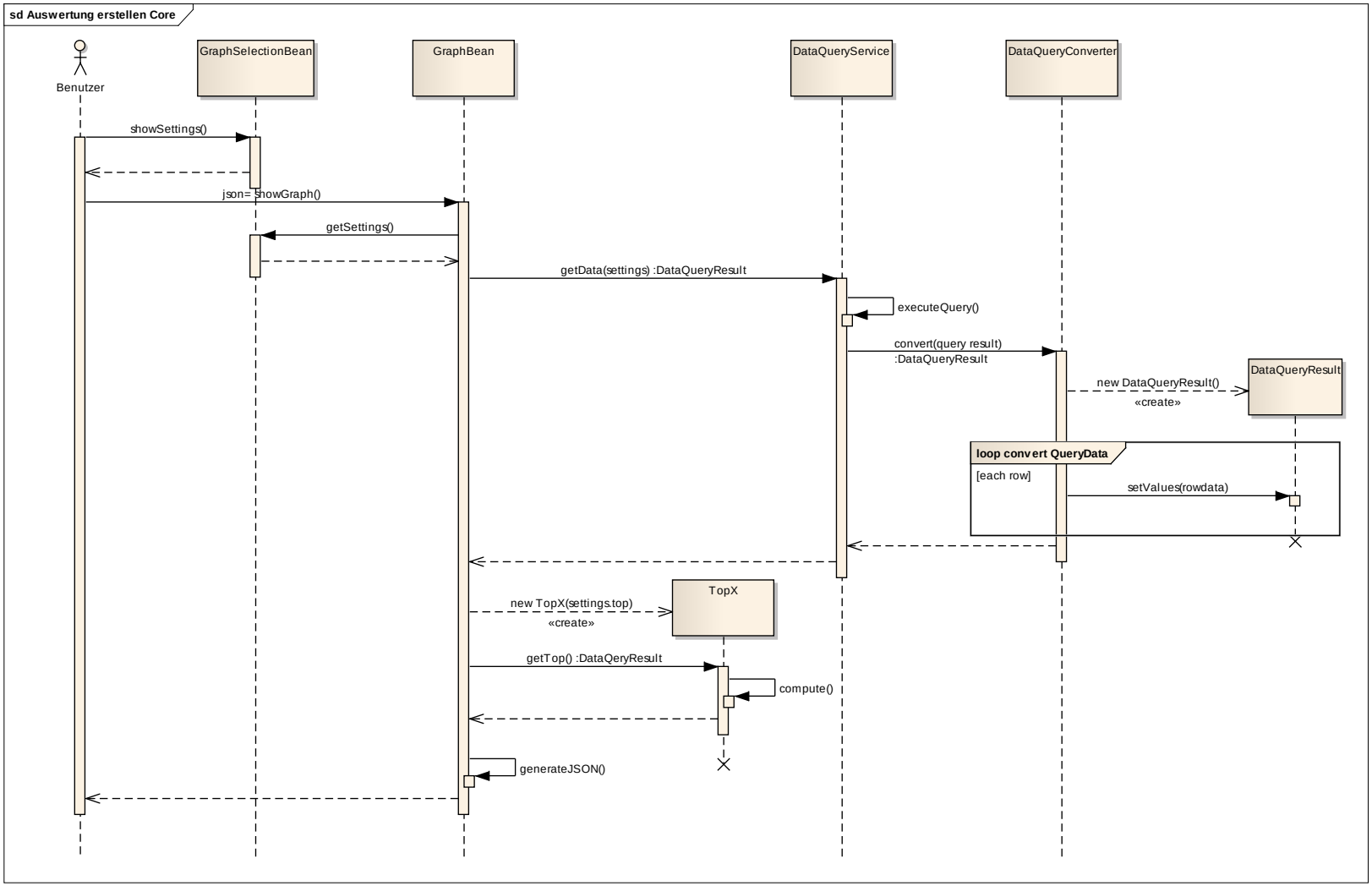


Abbildung 11: Sequenzdiagram Coreauswertung erstellen

3.4.3 Historisierung

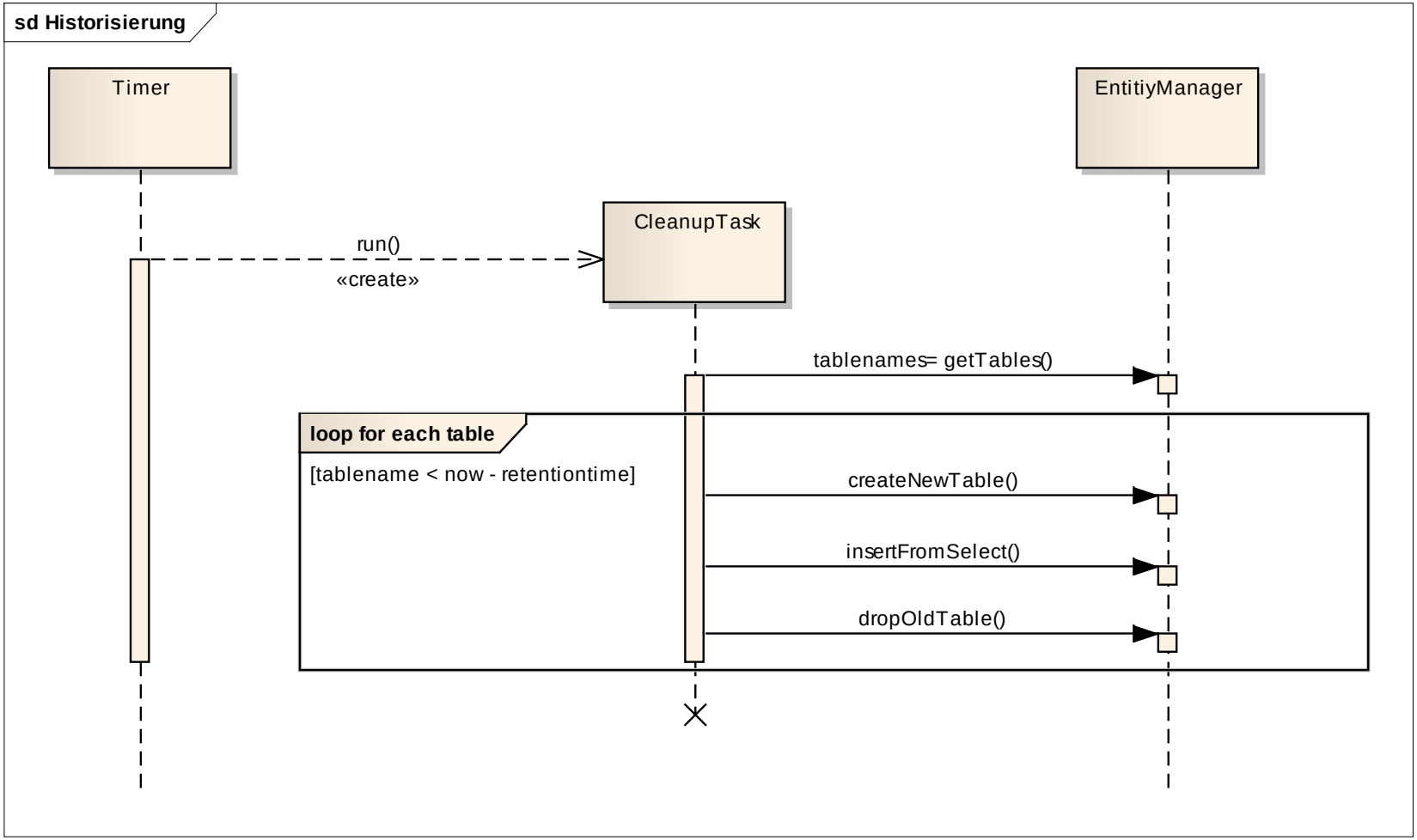


Abbildung 12: Sequenzdiagram Historisierung

4. Softwarearchitektur

4.1 Einführung

In diesem Dokument wird die gesamte Softwarearchitektur abgebildet und dokumentiert. Dies zeigt, wie die einzelnen Komponenten miteinander interagieren und wozu sie im System gebraucht werden.

4.2 Systemübersicht

Die folgende Graphik trägt alle Komponenten auf einem Bild zusammen. Pfeile zeigen zusätzlich die Interaktionsmöglichkeiten der verschiedenen Teile des Systems. Dabei ist auch darauf zu achten, dass gewisse Pfeile unidirektional und andere bidirektional sind. Das Produkt „PeeringBox“ besteht aus diversen Systemkomponenten, welche innerhalb der Systemgrenze abgebildet sind. In der Graphik sind nur die kommunizierenden Komponenten erwähnt.

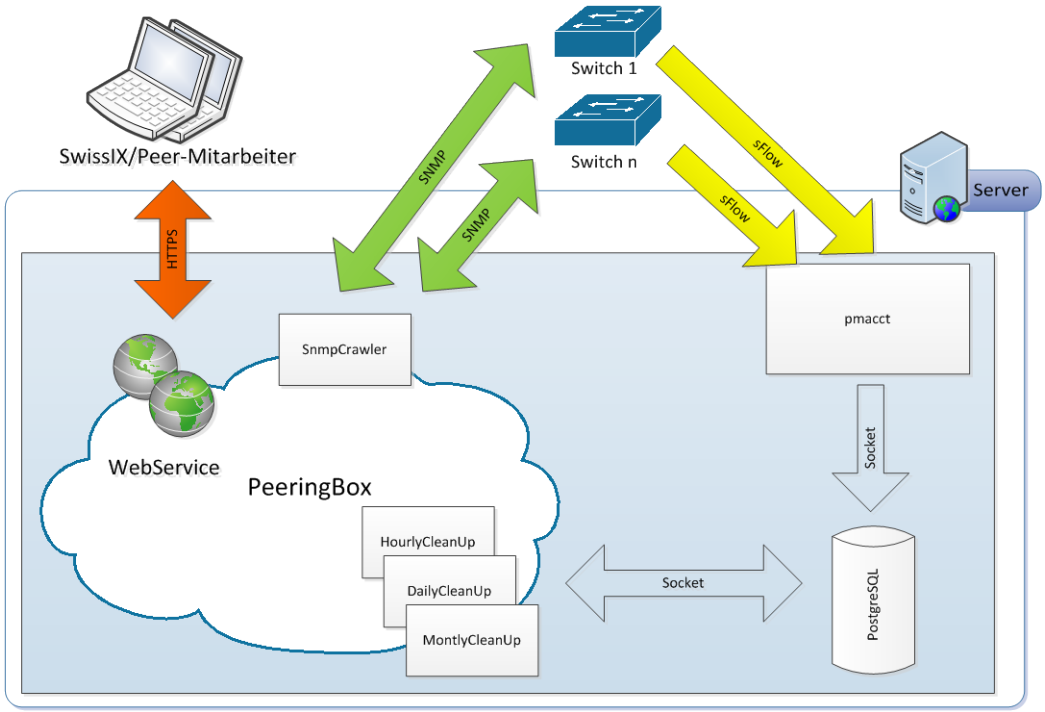


Abbildung 13: Systemübersicht

Nachfolgend werden diese Komponenten einzeln beschrieben. Dabei ist zu beachten, dass hier lediglich ein Überblick geboten werden soll. Die Genaue Funktionsweise wird unter 5. Implementation/Umsetzung beschrieben

4.2.1 Server

Der Server besteht aktuell aus einem physikalischen Gerät. Es wäre jedoch auch möglich die logischen Komponenten physikalisch zu trennen. Dieser Server wird von der SwissIX zur Verfügung gestellt und befindet sich in einem Rechenzentrum in der Schweiz.

4.2.2 WebService

Der Webservice gewährleistet die Interaktion des Menschen mit den verschiedenen Komponenten. Zum Einsatz kommt hier ein GlassFish-Server, der alle http(s)-Verbindungen terminiert und verwaltet.

4.2.3 SNMP-Crawler

Diese Komponente repräsentiert den Dienst, der in periodischen Abständen alle erfassten Switches abarbeitet. Dabei werden Informationen via SNMP abgeholt und in der Datenbank persistiert. Dazu gehören Interfacebezeichnungen (Description und Alias), sowie Einträge aus der Mac-Adress-Tabelle, womit ein Mapping der Mac-Adressen zu den verschiedenen Peers ermöglicht wird.

4.2.4 CleanUp-Tasks

Ein CleanUp-Task archiviert Datensätze, welche in einer feinen Granularität vorhanden sind in eine neue Tabelle welche eine gröbere Granularität besitzt. Dieser Schritt gewährleistet, dass die Datenmenge welche gespeichert wird nur um die jährlich anfallenden Daten steigt. Diese Tasks laufen ebenfalls in periodischen Abständen ab. Für jede Granularität ist ein eigener Task ablaufbereit, der lediglich zwischen zwei Tabellen vermittelt, wobei er von der Tabelle mit feiner Granularität in die Tabelle mit grober Granularität zusammenfasst. Nach diesem Vorgang werden die alten Daten gelöscht um Duplikate im System zu vermeiden.

4.2.5 pmacct

sFlow-Daten, welche von den verschiedenen Switches an den Server gesandt werden, müssen entgegengenommen, verarbeitet und persistiert werden. Diese Aufgabe übernimmt die Open-Source Anwendung pmacct. Die Anwendung wird mit Hilfe von Konfigurationsdateien anhand der benötigten Anforderungen konfiguriert.

4.2.6 PostgreSQL

Die Datenbank stellt die Zentrale Schnittstelle aller Komponenten dar. In der Datenbank werden gesammelte Daten gespeichert und von anderen Komponenten abgefragt und ausgewertet. Zum Einsatz kommt hier die Open-Source Anwendung PostgreSQL, welche in einem Funktionsvergleich als bestes Datenbanksystem für unsere Zwecke ausgewählt wurde.

4.3 Modularisierbarkeit

Die PeeringBox ist darauf ausgelegt, dass die Verschiedenen Komponenten auf einem Verteilten System lauffähig sind. Somit ist die Skalierbarkeit gewährleistet. Die Anwendung kann in drei verschiedene Bereiche eingeteilt werden, welche selbständig Operieren können: Datenbank, sFlow Collector (pmacct), Anwendungsserver.

Für die Operabilität muss der Zugriff von den sFlow Collectors und dem Anwendungsserver auf die Datenbank gewährleistet sein.

4.3.1 Datenbank

Da wir als Datenbank einen PostgreSQL Server verwenden, gibt es die Möglichkeit ein Server Clustering einzusetzen. Es gibt verschiedene Lösungen ein Clustering[3] bei PostgreSQL zu implementieren. Dies würde je nach eingesetztem Clustering ein Geschwindigkeitsvorteil und Ausfallsicherheit mit sich bringen.

4.3.2 sFlow Collector

Da jeder Switch einen eigenen pmacct Prozess besitzt, kann jeder Prozess selbständig auf einem eigenen Server laufen. Es müssen dazu lediglich die Verbindungsparameter der Datenbank in der pmacct Konfiguration angepasst werden.

4.3.3 Anwendungsserver

Der Anwendungsserver beherbergt den Webserver und die Businesslogik der gesamten Anwendung. Es gibt auch beim GlassFish die Möglichkeit des Clustering[4]. Dies bringt Lastverteilung und Ausfallssicherheit mit sich.

[3] http://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling
[4] https://glassfish.java.net/public/clustering31.html

4.4 Architektonische Ziele & Einschränkungen

4.4.1 Ziele

4.4.1.1 Erweiterbarkeit

Die Arbeit ist lediglich auf den Scope der SwissIX ausgelegt. Da jedoch daraus ein Produkt entstehen soll, das auch bei anderen, nationalen, wie auch internationalen IX eingesetzt werden könnte, soll das System so aufgebaut werden, dass mit wenig Aufwand eine Anpassung an andere Gegebenheiten möglich wird. Dies beinhaltet:

- Andere Netzwerkgeräte (andere Hersteller, sowie andere Modelle)
- Andere Graphen (Libraries, wie auch Auswertungen)
- Andere sFlow-Attribute

4.4.1.2 Usability

Da die Interaktion lediglich im Webinterface stattfindet, wird viel Wert darauf gelegt, dass eine einfache Bedienung ermöglicht wird. Die Menüstruktur soll eindeutig und logisch sein. Die Optionen auf der Seite sollen klar beschrieben sein und es soll zu jedem Zeitpunkt klar sein, wo man sich befindet und welche Auswertungen man gerade vor Augen hat.

4.4.1.3 Privacy und Security

Die Applikation sammelt Daten, welche für die teilnehmenden Peers teils einen hohen Wert haben und welche sie zu schützen versuchen. Daten die den eigenen Peer betreffen, werden gerne eingesehen, weil sich dadurch ein allfälliger Kostenpunkt auf der Seite der Peers reduziert oder gar erübrigt. Dabei muss aber zu jedem Zeitpunkt sichergestellt werden, dass niemand die Daten eines anderen Peers einsehen kann.

Im Weiteren werden Benutzerdaten, wie E-Mail und Passwor, in der Datenbank gespeichert. Da heutzutage viele Benutzer immer dieselbe E-Mail-Adresse und dasselbe Passwort verwenden, sind wir dazu verpflichtet diese Daten sensitiv zu behandeln. Diese Daten werden lediglich den Mitarbeitern des IX offen gelegt. Passwörter werden nur als Hash gespeichert.

4.4.2 Einschränkungen

Das System arbeitet grösstenteils vollautomatisch und garantiert so eine kleinere Fehlerquelle. Folgende Einschränkungen sind an die Automatik gegeben:

- Switches müssen im System manuell erfasst werden.
- Benutzer müssen manuell erfasst werden.
- Die Zuordnung von Benutzern an die entsprechenden Peers müssen manuell vorgenommen werden.
- Rechte müssen dem Benutzer manuell vergeben werden. Im Normalfall befindet sich ein neuer Benutzer in der Gruppe „Users“.

4.5 Externes Design (Webseite)

Während der Designanalyse wurden einige Tage dafür investiert, konkrete Vorstellungen über das graphische Userinterface zu erlangen. Dabei wurden sogenannte Mockups erstellt, welche die Handhabung von Funktionalität, die Positionierung der Elemente und die Designvorstellungen zeigen.

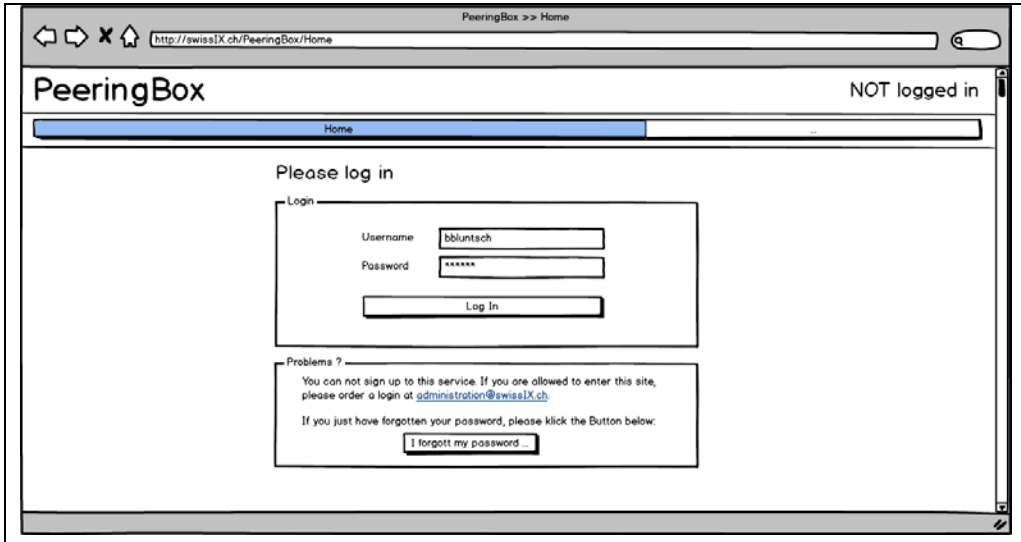


Abbildung 14: Mockup Welcome Page

Welcome Page

Im ausgeloggten Zustand erscheint eine Welcome-Seite, die einem das Login anbietet. Zusätzlich erhält man jeweils die Möglichkeit das Passwort wiederherzustellen, falls der Benutzer sein eigenes Passwort vergessen haben sollte.

Abbildung 15: Mockup Welcome Page intern

Interne Welcome-Seite

Sobald man sich im internen Bereich befindet, wird die Menüstruktur geladen. Erst jetzt hat man Zugriff auf die Funktionalität des Produktes PeeringBox. Administratoren haben eine erweiterte Menüstruktur gegenüber „normalen“ Benutzern.

Im Weiteren werden hier interne Informationen bekannt gegeben, sowie Informationen über den eigenen Benutzer angezeigt.

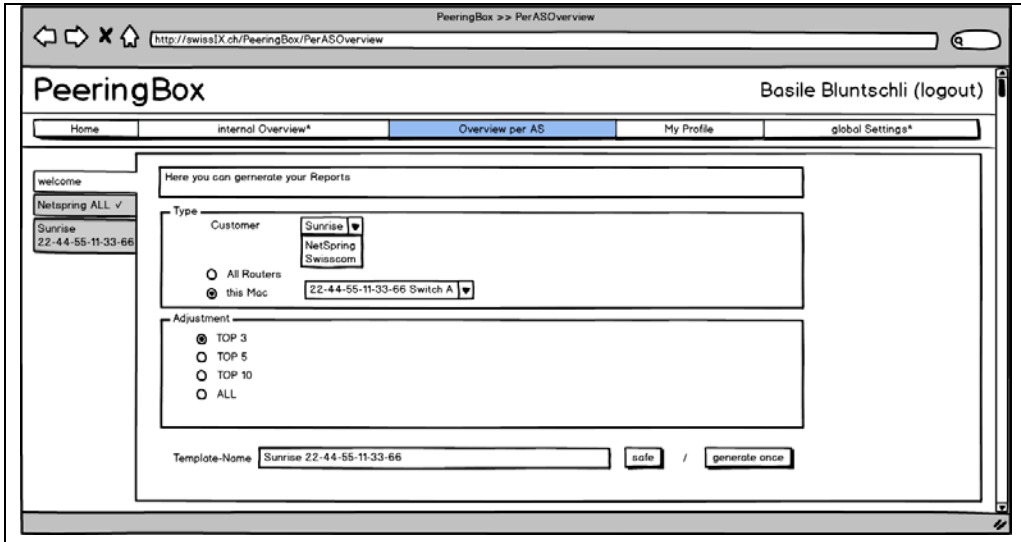


Abbildung 16: Mockup Overview per AS

Overview per AS

Hier ist nun eine Konfigurationsseite ersichtlich, womit man Einstellungen treffen kann, um einen gewünschten Graphen zu erhalten. Sämtliche Kundennamen und Macadressen sind hierbei rein fiktiv.

Die getroffenen Einstellungen können gespeichert werden, wodurch sie auf der rechten Seite eingeblendet werden.

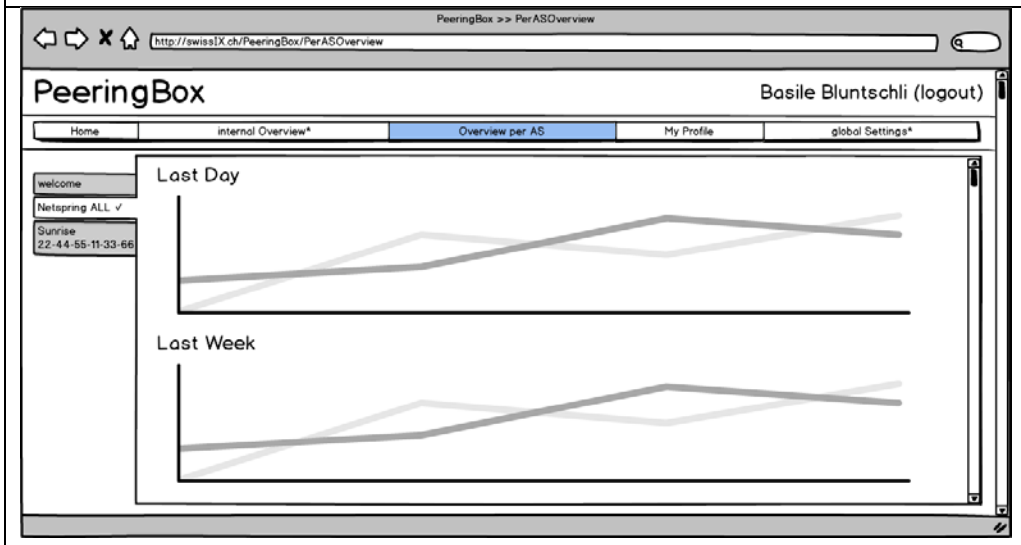


Abbildung 17: Mockup Graph View

Graph View

Hier werden die errechneten Graphen demonstriert. Diese Graphen werden in unterschiedlichen Perioden separiert.

Durch einen Klick auf den entsprechenden Graphen werden Details ersichtlich.

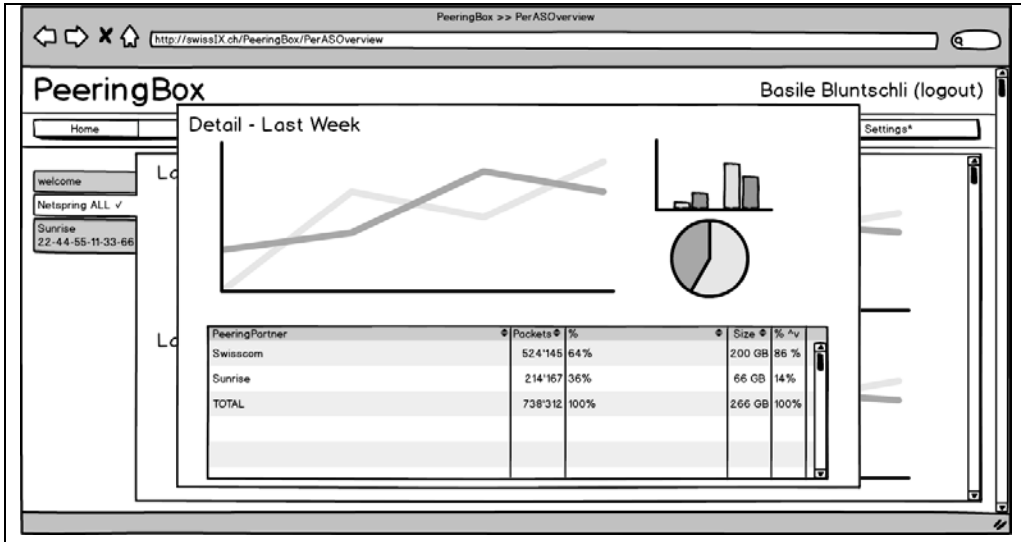


Abbildung 18: Mockup Graph View Detail

Graph View - Detail

Links ist nun die schematische Darstellung einer detaillierten Auswertung zu sehen.

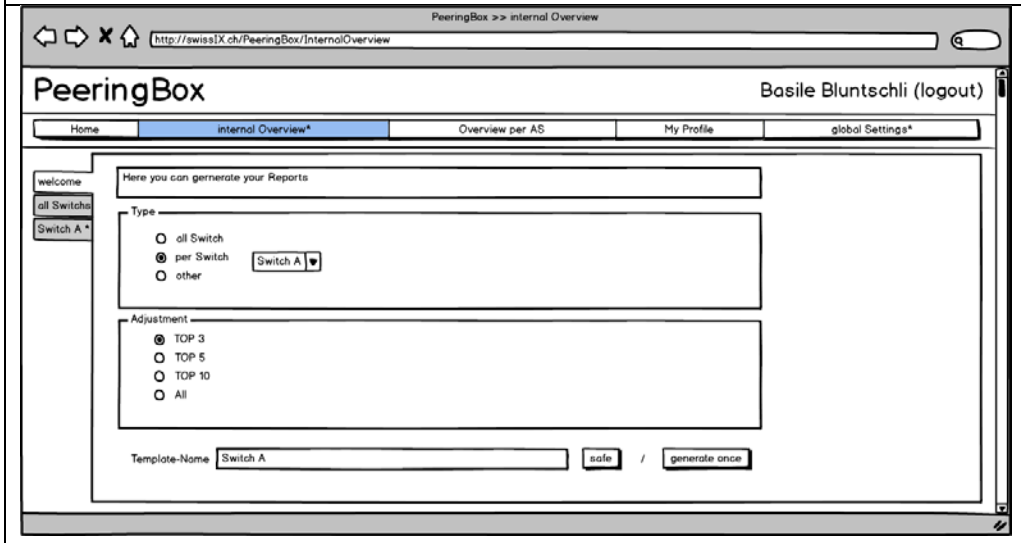


Abbildung 19: Mockup Internal Overview

Internal Overview

Für Mitarbeiter des IX werden ähnliche Elemente vorgesehen, nur sind andere, IX-bezogene Einstellungen, möglich. So lässt sich hier der entsprechende Port des Switches wählen.

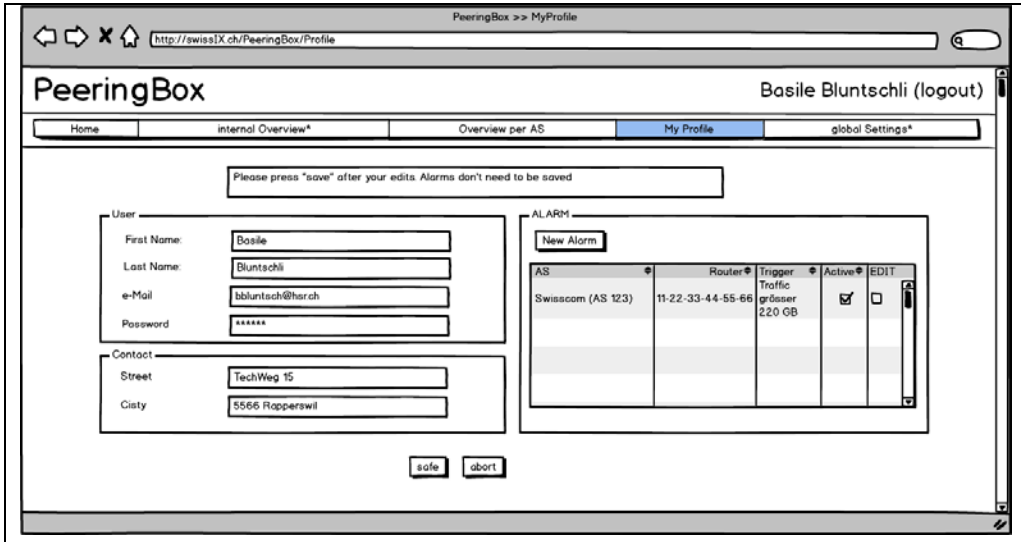


Abbildung 20: Mockup My Profile

My Profile

Im eigenen Profil können Einstellungen zur Person, bzw. dem Login vorgenommen werden.

Zusätzlich wird hier die Möglichkeit geboten, die eigenen Alarme anzuzeigen, zu bearbeiten und neue Alarme zu erfassen.

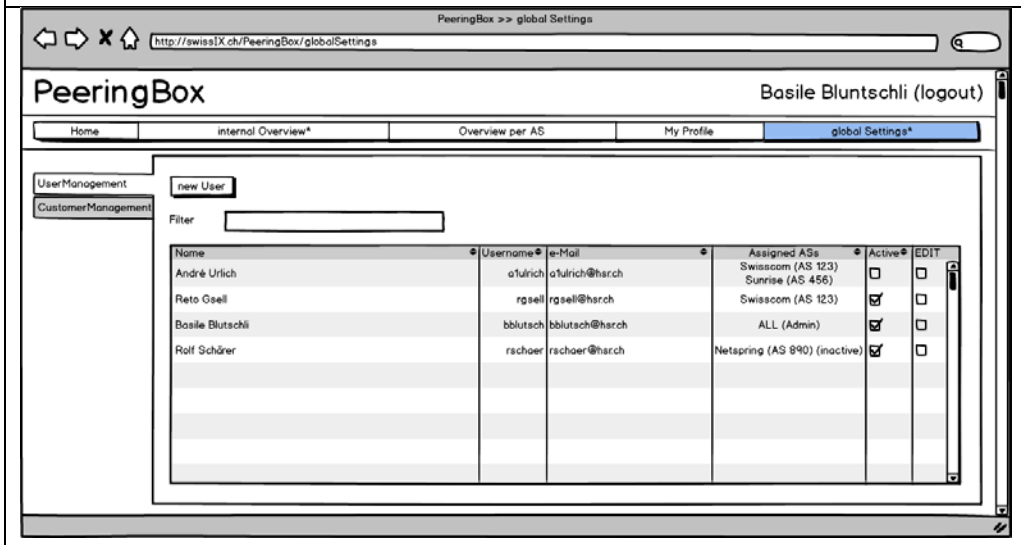


Abbildung 21: Mockup User Management

User Management

Als Administrator hat man die Möglichkeit, sämtliche Benutzer anzeigen zu lassen und zu editieren.

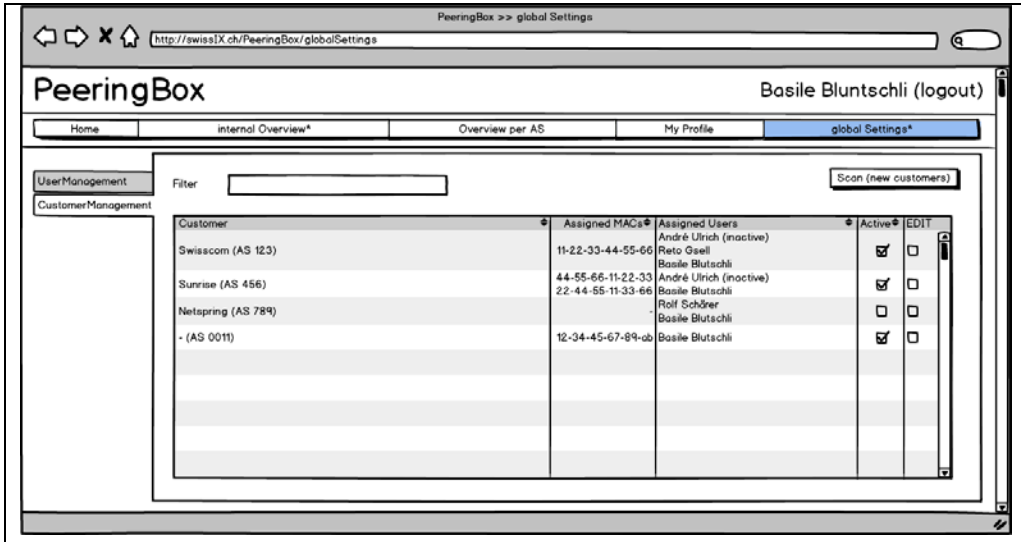


Abbildung 22: Mockup Peer Management

Peer Management

Es können sämtliche Peers aufgelistet und Zuweisungen vorgenommen werden.

5. Implementation/Umsetzung

5.1 Datenbank

5.1.1 Varianten

5.1.1.1 Variante 1: - NoSQL

Eine NoSQL Datenbank bietet im Gegensatz zu üblichen relationalen Datenbanksystemen eine bessere horizontale Skalierung und eine höhere Verfügbarkeit. Dabei eignet sich so ein System für hohe Schreib- und Lesezugriffe. Diese Vorteile bringen jedoch den Nachteil einer schlechteren Konsistenz der Daten mit sich. Vor allem bei Daten, welche nicht relational zueinander stehen, ist eine NoSQL Datenbank sinnvoll.

5.1.1.1.1 Vorteile

- Hohe Performance für Schreib- und Lesezugriffe
- Gute Skalierbarkeit (vor allem bei Benutzung in Clustern)
- Flache Datenstruktur

5.1.1.1.2 Nachteile

- Persistierung der Anwendungsdaten sehr aufwendig
- Relation von historisierten Daten und Anwendungsdaten schwierig

5.1.1.2 Variante 2: - MySQL

MySQL ist das weltweit am weitesten verbreitete relationale Datenbanksystem. Es ist als Open-Source wie auch als kommerzielle Enterpriseversion für verschiedene Betriebssysteme verfügbar und wird von Oracle vertrieben.

5.1.1.2.1 Vorteile

- Stabilität
- Skalierbarkeit
- Enormer Funktionsumfang
- Pmacct Plugin vorhanden

5.1.1.2.2 Nachteile

- Administration nur über Datenbank möglich (nicht über Shell)
- Vertrieb durch kommerzielle Firma
- Backup im laufenden Betrieb ist zum Beispiel nur mit der Enterpriseversion möglich

5.1.1.3 Variante 3 - PostgreSQL

PostgreSQL ist ein objektrelationales Datenbanksystem. Es implementiert die Speicherungen von nicht atomaren Daten, Vererbung und Objektidentitäten. Zusätzlich können selbstdefinierte Datentypen, Operatoren und Funktionen erfasst werden. Das System bietet ein fortschrittliches Transaktionsmanagement und unterstützt referentielle Integrität.

5.1.1.3.1 Vorteile

- Wird in den Modulen Dbs1 und Dbs2 an der HSR unterrichtet
- Reine Open-Source-Lösung
- Umfassendes Transaktionskonzept, das Multiversion Concurrency Control (MVCC) unterstützt
- Mengenoperationen
- Maximale Datenbankgrösse nur durch zur Verfügung stehenden Speicher begrenzt
- Views, die mit Hilfe von Regeln (Rules und Triggers) auch schreibfähig sein können (Updatable Views für Historisierung nützlich)
- Prozeduren (stored procedures) sind in verschiedenen Sprachen möglich
- Schnittstellen zu verschiedenen Programmiersprachen
- Erweiterbarkeit durch Funktionen, selbstdefinierbaren Datentypen und Operatoren
- Bessere Performance bei vielen Schreibzugriffen als MySQL
- Pmacct Plugin vorhanden
- Tablespaces möglich (Physikalische Trennung von historisierten Daten und Anwendungsdaten)

5.1.1.3.2 Nachteile

- Keinen Support (nur durch Community)

5.1.2 Entscheid und Begründung

Die Entscheidung fällt auf: Variante 3 - PostgreSQL

In erster Linie unterscheiden wir zwischen relationalen und nicht relationalen Datenbanken. Da wir im Umfang des Produktes „PeeringBox“ Daten relational zueinander verwalten müssen (vor allem im Anwendungsbereich), wäre eine NoSQL Datenbank eher ungünstig. Die Verwendung einer NoSQL Datenbank würde dazu führen, dass zwei verschiedene Datenbanken benötigt werden, was die Verwaltung verkomplizieren würde. Zudem wird vom sFlow Collector Pmacct nur MySQL und PostgreSQL mit einem Datenbankplugin unterstützt.

Der Entscheid auf die PosgreSQL Datenbank wurde hauptsächlich durch die bessere Performance bei vielen Schreibzugriffen (gegenüber von MySQL) gefällt. Zusätzlich hat das Projektteam schon Erfahrungen mit dem Datenbanksystem aus den Modulen Dbs1 und Dbs2.

5.1.3 Datenbankmodell

Auf Grund des Klassendiagramm wurde ein Datenbankmodell erstellt, das alle Objekte abbilden kann. Die Striche zwischen den verschiedenen Tabellen zeigen Beziehungen an. Hier ist jedoch ausdrücklich zu erwähnen, dass zwischen der Tabelle „Port_allocation“ und der einzelnen Samples keine richtige Beziehung besteht. Diese Beziehung wird durch die Applikation während der Laufzeit durch die Erstellung entsprechender Queries erstellt.

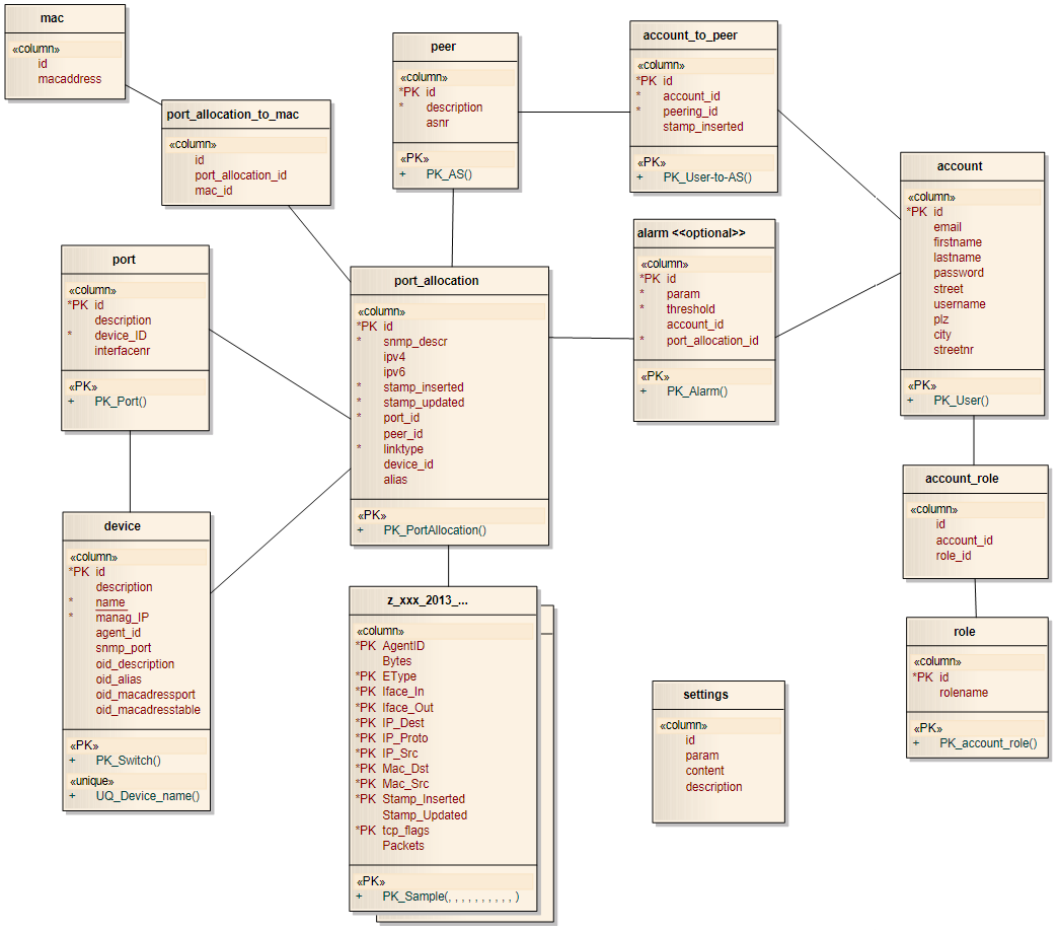


Abbildung 23: Datenbankmodell

5.1.4 Vererbung und Index

5.1.4.1 Table Inheritance

Vererbung in einem Datenbanksystem bringt einen enormen Vorteil mit sich. Ein enormer Vorteil besteht darin, dass Abfragen auf die Basistabelle getätigt werden können. Dies löst automatisch eine Suche in all den erbenden Tabellen aus. Des Weiteren können Tabellen Attribute und Indexe von der Basistabelle erben.

Im konkreten Falle der Bachelorarbeit sieht die Basistabelle wie folgt aus. Sie enthält alle Attribute, die alle Tabellen erhalten sollen.

```
CREATE TABLE IF NOT EXISTS z_base(
        agent_id        BIGINT                          NOT NULL DEFAULT 0,
        iface_in        BIGINT                          NOT NULL DEFAULT 0,
        iface_out       BIGINT                          NOT NULL DEFAULT 0,
        etype           CHAR(5)                         NOT NULL DEFAULT '',
        mac_src         macaddr                         NOT NULL DEFAULT '0:0:0:0:0:0',
        mac_dst         macaddr                         NOT NULL DEFAULT '0:0:0:0:0:0',
        port_src        INT                             NOT NULL DEFAULT 0,
        port_dst        INT                             NOT NULL DEFAULT 0,
        ip_proto        SMALLINT                        NOT NULL DEFAULT 0,
        tcp_flags       SMALLINT                        NOT NULL DEFAULT 0,
        packets         BIGINT                          NOT NULL,
        bytes           BIGINT                          NOT NULL,
        stamp_inserted  timestamp without time zone NOT NULL DEFAULT '0001-01-01 00:00:00',
        stamp_updated   timestamp without time zone
);
```

Das folgende Statement zeigt die Erstellung einer Tabelle, die von der Tabelle „z_base“ sämtliche Attribute und Indexe erbt und dabei automatisch bei der Basistabelle als Kindtabelle eingetragen wird.

```
--CREATE TABLE IF NOT EXISTS z_day_wxyz_ab_cd(
--        id              BIGSERIAL                       PRIMARY KEY
--)INHERITS (z_base);
```

Der Primärschlüssel wird bewusst nur in der Kindtabelle erzeugt. Ein Primärschlüssel wird stets von der Datenbank generiert. Falls die Kindtabellen diesen ebenfalls erben würden, müssten sich alle Tabellen ihre nächste freie ID von der Basetable holen. Da im Tag 5 bis 20 Millionen Records geschrieben werden, ist dies unerwünscht. Des Weiteren dient es der Fehlervermeidung, weil somit das Problem von duplizierten Einträgen behoben wird.

5.1.4.2 Index

Indexe katalogisieren Tabellen nach den Inhalten der spezifizierten Attribute. Eine Abfrage nach Attributen wird dann lediglich im dafür optimierten Index ausgeführt. Dies ist um ein Vielfaches schneller, als wenn die ganze Tabelle durchsucht werden muss.

Der untenstehende Code zeigt, wie diese Indexe in der Anwendung realisiert sind. Alle Indexe werden direkt auf der Basistabelle definiert und werden durch die Vererbung direkt an die Kindtabellen weiter vererbt. Somit ist auch sichergestellt, dass alle Kindtabellen die entsprechenden Indexe besitzen.

```
CREATE INDEX z_base_agent_id_iface_in_stamp_inserted_idx
  ON z_base
  USING btree
  (agent_id, iface_in, stamp_inserted);

CREATE INDEX z_base_agent_id_iface_out_stamp_inserted_idx
  ON z_base
  USING btree
  (agent_id, iface_out, stamp_inserted);

CREATE INDEX z_base_agent_id_mac_dst_stamp_inserted_idx
  ON z_base
  USING btree
  (agent_id, mac_dst, stamp_inserted);

CREATE INDEX z_base_agent_id_mac_src_stamp_inserted_idx
  ON z_base
  USING btree
  (agent_id, mac_src, stamp_inserted);
```

Die Indexe sind auf die verwendeten Queries angepasst worden. Je spezifischer ein Index definiert ist, desto optimierter arbeitet er. Ein „OR“ in der Query veranlasst die Datenbank, zwei separate Sub Queries zu schreiben. Um unsere Abfragen zu optimieren, musste analysiert werden, welche endgültigen Queries aus den Abfragen resultieren. Um dies zu verdeutlichen, hier ein Auszug eines Queries.

```
WHERE   agent_id = 1
        AND (   iface_in = 199
                OR iface_out=199)
```

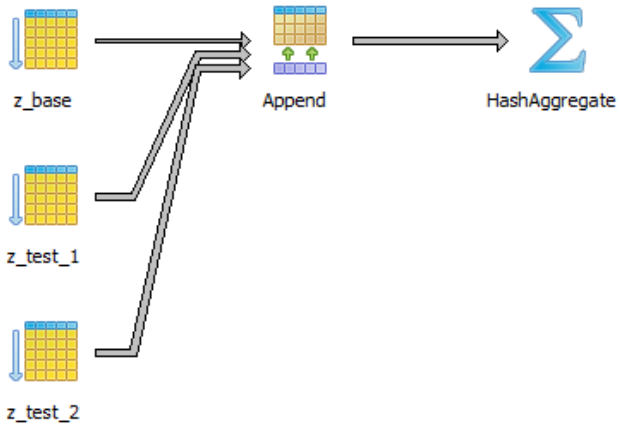
Das obenstehende Query wird in das Untenstehende konvertiert.

```
WHERE   agent_id = 1 AND iface_in = 199
        OR
        agent_id = 1 AND iface_out=199
```

Pro Unterabfrage bleibt somit das Attribut „agent_id“ und je nach dem „iface_in“ oder iface_out“ bestehen. Dazu kommt stets „stamp_inserted“, da danach zusätzlich gruppiert wird. Daraus resultiert aus dem Beispiel von oben ein Index mit den Attributen agent_id, iface_in und stamp_inserted.

5.1.4.3 Beispiel Query via Vererbung

Die folgenden Bilder zeigen den Ablauf einer Abfrage auf der Basistabelle. Wie zu sehen ist, werden die Basistabelle, inkl. aller Kindtabellen durchsucht und die Resultate aneinander gehängt.

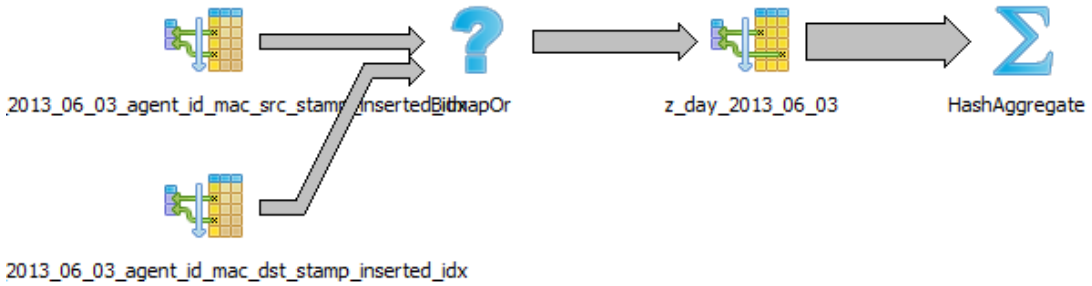


	QUERY PLAN text
1	HashAggregate (cost=39.99..40.02 rows=3 width=40) (actual time=0.013..0.013 rows=0 loops=1)
2	-> Append (cost=0.00..39.95 rows=3 width=40) (actual time=0.010..0.010 rows=0 loops=1)
3	-> Seq Scan on z base (cost=0.00..0.00 rows=1 width=40) (actual time=0.002..0.002 rows=0 loops=1)
4	Filter: ((agent id = 1) AND ((iface in = 199) OR (iface out = 199)))
5	-> Seq Scan on z test 1 z base (cost=0.00..19.98 rows=1 width=40) (actual time=0.001..0.001 rows=0 loops=1)
6	Filter: ((agent id = 1) AND ((iface in = 199) OR (iface out = 199)))
7	-> Seq Scan on z test 2 z base (cost=0.00..19.98 rows=1 width=40) (actual time=0.001..0.001 rows=0 loops=1)
8	Filter: ((agent id = 1) AND ((iface in = 199) OR (iface out = 199)))
9	Total runtime: 0.116 ms

Abbildung 24: Vererbung

5.1.4.4 Beispiel Query via Index

Diese Bilder beschrieben die optimierte Abfrage auf dem Index. Durch die Verwendung eines OR, werden dabei zwei Abfragen erstellt, die auf unterschiedlichen, eigenes dafür erstellten Indexes ausgeführt werden.



	QUERY PLAN text
1	HashAggregate (cost=101558.83..101593.69 rows=3486 width=36) (actual time=277.944..281.577 rows=6696 loops=1)
2	-> Bitmap Heap Scan on z_day_2013_05_20 (cost=1069.83..101123.11 rows=34857 width=36) (actual time=157.290..203.575 rows=48621 loops=1)
3	Recheck Cond: (((agent_id = 1) AND (mac_src = '00:1c:0f:5f:80:00'::macaddr)) OR ((agent_id = 1) AND (mac_dst = '00:1c:0f:5f:80:00'::macaddr)))
4	-> BitmapOr (cost=1069.83..1069.83 rows=34889 width=0) (actual time=151.458..151.458 rows=0 loops=1)
5	-> Bitmap Index Scan on z_day_2013_05_20_agent_id_mac_src_stamp_inserted_idx (cost=0.00..563.22 rows=18746 width=0) (actual time=80.522..80.522 rows=24544 loops=1)
6	Index Cond: ((agent_id = 1) AND (mac_src = '00:1c:0f:5f:80:00'::macaddr))
7	-> Bitmap Index Scan on z_day_2013_05_20_agent_id_mac_dst_stamp_inserted_idx (cost=0.00..489.18 rows=16142 width=0) (actual time=70.933..70.933 rows=24077 loops=1)
8	Index Cond: ((agent_id = 1) AND (mac_dst = '00:1c:0f:5f:80:00'::macaddr))
9	Total runtime: 282.242 ms

Abbildung 25: Index

5.1.5 Zugriff

Objekte, ausgenommen Samples, werden bei Verwendung aus der Datenbank geladen und interpretiert. Innerhalb der Applikation wird nun mit diesen Objekten gearbeitet. Bei Veränderungen der Objekte werden diese automatisch in der Datenbank nachgetragen. Diese Funktionalität wird mit Hilfe der Java Persistence API (JPA) sichergestellt. Samples werden direkt per Query ausgewertet. Durch diese Teilung der Mechanismen konnte ein optimaler Mittelweg zwischen Funktionalität und Performance gefunden werden.

5.1.5.1 JPA

Um mit JPA arbeiten zu können muss man auf einen sogenannten OR-Mapper zurückgreifen. Dieser ist dafür verantwortlich, dass die interne Struktur des Objektes richtig zwischen Datenbank und Java-Anwendung verarbeitet wird. Es gibt eine Vielzahl von Implementationen, die eine solche Funktionalität bieten. Die bekanntesten sind: DataNucleus, EclipseLink, Hibernate, OpenJPA und ObjectDB.

Wir haben uns aus den folgenden Gründen für die Verwendung von Eclipse Link entschieden:

- Es handelt sich dabei um die Referenzimplementierung, wodurch wir sicher gehen können, dass alle benötigten Funktionen das erwartete Verhalten an den Tag legen.
- Durch die Verwendung von Eclipse als IDE liegt die Verwendung von Eclipse Link nahe, da entsprechende Hilfsfunktionen zur Verfügung stehen.
- Im direkten Vergleich schliesst Eclipse Link super im Zusammenspiel mit PostgreSQL ab. Da wir uns bereits für diese Datenbank entschieden haben, ein wichtiger Pluspunkt.
    - Vergleiche:
        - http://www.jpab.org/All/All/All.html
        - http://www.jpab.org/EclipseLink.html

5.1.5.2 Query

Samples können unmöglich über JPA ausgewertet werden, denn wenn jedes Sample in der Applikation durchsucht werden müsste, würde das bedeuten, dass je nach Datenmenge gut 100 GB abgefragt, deserialisiert und durchsucht werden müssten. Da dies die Ressourcen eines normalen Servers bei weitem übersteigt, wird dieser Teil dem Datenbanksystem überlassen. Je nach Selektion der Auswertungsmöglichkeiten wird ein Query zusammengestellt, das lediglich die benötigten Daten liefert.

5.2 Anwendungsserver

Um eine „Java Enterprise Edition“-Anwendung (JEE) ausführen zu können, ist ein sogenannter Anwendungsserver von Nöten. Dieser stellt eine Schnittstelle zu diversen Komponenten zur Verfügung. Unter anderem einen Webserver und Mailservice zur Verfügung. Dies bringt enorme Vorteile, denn Änderungen an Komponenten werden automatisch in den verschiedenen Anwendungen, die innerhalb des Servers laufen, übernommen.

Auch hier gibt es eine Vielzahl von Implementationen, die genau diese Dienste bieten. Die wohl Bekanntesten sind Apache Geronimo, GlassFish, JBoss Application Server, JOnAS, Oracle Application, Server und WebSphere.

Die Wahl ist auf GlassFish gefallen. Hier einige der ausschlaggebenden Gründe:

- Open-Source – Somit fallen keine Lizenzkosten an und Änderungen am Code wären möglich.
- Es handelt sich bei GlassFish um die Referenzimplementierung eines Applikationsservers nach Spezifikation von Oracle.
- Grosse Community – Viele Fachartikel im Internet, die bei Problemen weiter helfen.
- Wird immer noch aktiv weiter entwickelt.

5.2.1 Funktionsweise

5.2.1.1 Ressourcen

5.2.1.1.1 JDBC

Der JDBC Connection Pool stellt dem GlassFish Server einen Pool zu verschiedenen Datenbanken zur Verfügung. Im Falle der PeeringBox wird eine JTA Ressource zur Verfügung gestellt. Diese Ressource wird mit dem korrekten Treiber und den richtigen Verbindungsdaten auf dem Server konfiguriert. Mit der Verwendung der JDBC Ressource ist die Datenbank von der Anwendung entkoppelt. Mit dieser Lösung muss sich der Programmierer nicht mehr um die Datenbank Anbindung kümmern und kann direkt auf die Ressource des Anwendungsservers zugreifen. Zusätzlich werden die Datenbanktransaktionen vom Server verwaltet, somit ist die Transaktionssicherheit bei mehreren, gleichzeitigen Zugriffen sichergestellt. Im Falle der PeeringBox werden die Transaktionen der Historisierung in der Anwendung verwaltet, da das Transaktionsmanagement des Servers sonst die Datenbank blockieren könnte.

5.2.1.1.2 Container Manager / JNDI

Der Container Manager des GlassFish Servers stellt Ressourcen (Java Beans) der JEE Anwendung zur Verfügung. Diese werden in der Applikation direkt über Annotations erstellt und stellen Services zur Verfügung. Diese Ressourcen können entweder über Ressource Injection oder durch einen Lookup abgerufen werden. Im Falle der Verwendung der Ressource in einem Container Managed Objekt (Java Bean) können die Ressourcen direkt vom Anwendungsserver Injected werden. Bei einem POJO (Plain Old Java Object) müssen Ressourcen über einen Lookup in den JNDI Ressourcen abgeholt werden.

5.2.1.1.3 JavaMail

Der GlassFish Server kann verschiedene JavaMail-Sessions verwalten. In diesen Sessions können die Verbindungsdaten zu einem Mailserver gespeichert werden. Die Mail Session ist somit von der Anwendung entkoppelt und kann bei Bedarf auch ersetzt werden. Es kann also einfach eine neuer Mailserver konfiguriert werden, ohne die Anwendung zu verändern.

5.2.1.2 HTTP-Service

Der HTTP-Service stellt den Webserver dar. Der HTTP-Service verwaltet die verschiedenen HTTP-Listener, welche einem Anwendungsport (zum Beispiel: Port 80/http) zugeordnet sind. Der HTTP-Service regelt das Verhalten der Requests an den Server. Im Falle der PeeringBox werden ein http- und ein HTTPS-Listener verwaltet.

5.2.1.3 Security

Der GlassFish Server kann verschiedene Security Realms verwalten. Die Accounts für die Benutzer von PeeringBox werden über einen Realm auf dem Server authentisiert. Im Authentisierungsrealm werden die Benutzer-/Rollentabelle aus der JDBC Ressource und der Digest-Algorithmus angegeben. Diese Lösung stellt eine systemweite Authentisierung zur Verfügung.

5.3 Frontend

PeeringBox setzt das MVC (Model View Controller) Pattern ein.

- Model: Business domain / Service layer (EJB/JPA/DAO)
- View: JSF Code (XHTML)
- Controller: FacesServlet

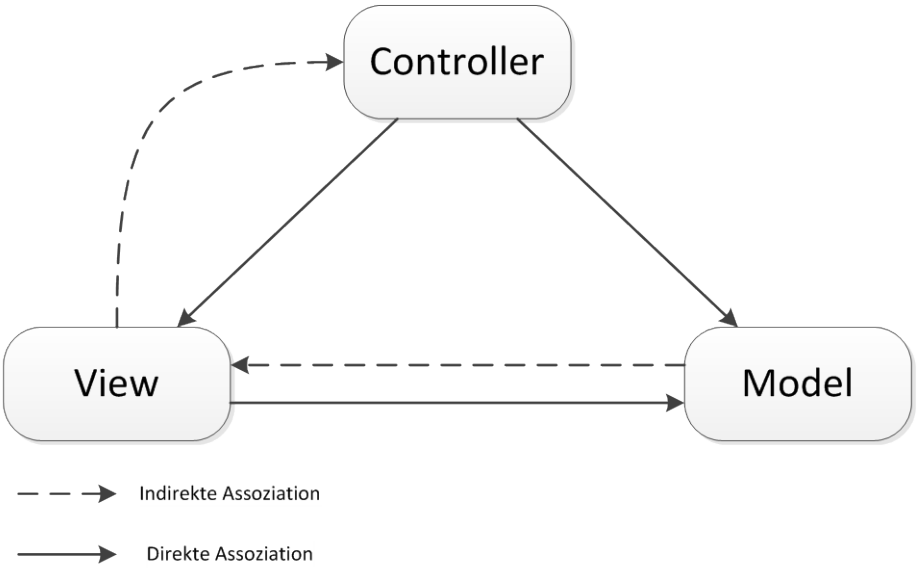


Abbildung 26: Model View Controller

Dieses Architekturpattern erlaubt es die Anwendung flexibel zu gestalten. Einzelne Komponenten werden dadurch wiederverwendbar. In unserem Fall kann die Businesslogik sowohl im Backend, wie auch im Frontend benutzt werden.

Das Model beinhaltet die Daten, welche dargestellt werden möchten. Die View stellt die Präsentationsschicht dar. Der Controller verwaltet die Views. Im Controller werden Benutzerinteraktionen entgegengenommen und ausgewertet. Der Controller entscheidet welche Daten im Model geändert werden müssen.

5.3.1 JSF

JSF (Java Server Faces) ist ein Framework zur Entwicklung von Webapplikationen mit Java. JSF setzt eine Java EE Applikation voraus. Mit JSF können einfache Webapplikationen entwickelt werden. Die View wird dabei in XHTML erstellt. JSF generiert daraus den nötigen HTML Code welcher an den Client gesendet wird. PeeringBox verwendet die JSF Referenz-Implementation Mojarra von Oracle.

5.3.2 PrimeFaces

PrimeFaces[5] ist eine Open-Source Komponenten-Library die auf JSF basiert. PrimeFaces bietet eine grosse Anzahl von GUI Komponenten[6]. PrimeFaces ist eine relativ neue Library. Folgender Google-Trend Graph zeigt die Popularität von PrimeFaces im Vergleich zu seinen Konkurrenten.

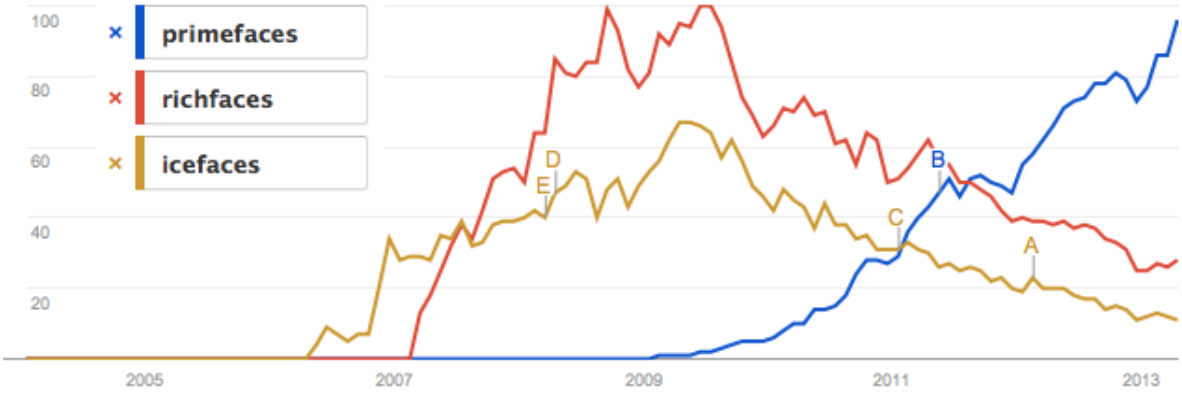


Abbildung 27: Trend PrimeFaces

[5] http://www.primefaces.org/
[6] http://www.primefaces.org/showcase/ui/home.jsf

5.3.3 Eindruck

Hier ein paar Ausschnitte, die einen ersten Eindruck ermöglichen sollen. Detailliertere Angaben sind dem Benutzerhandbuch im Anhang zu entnehmen.

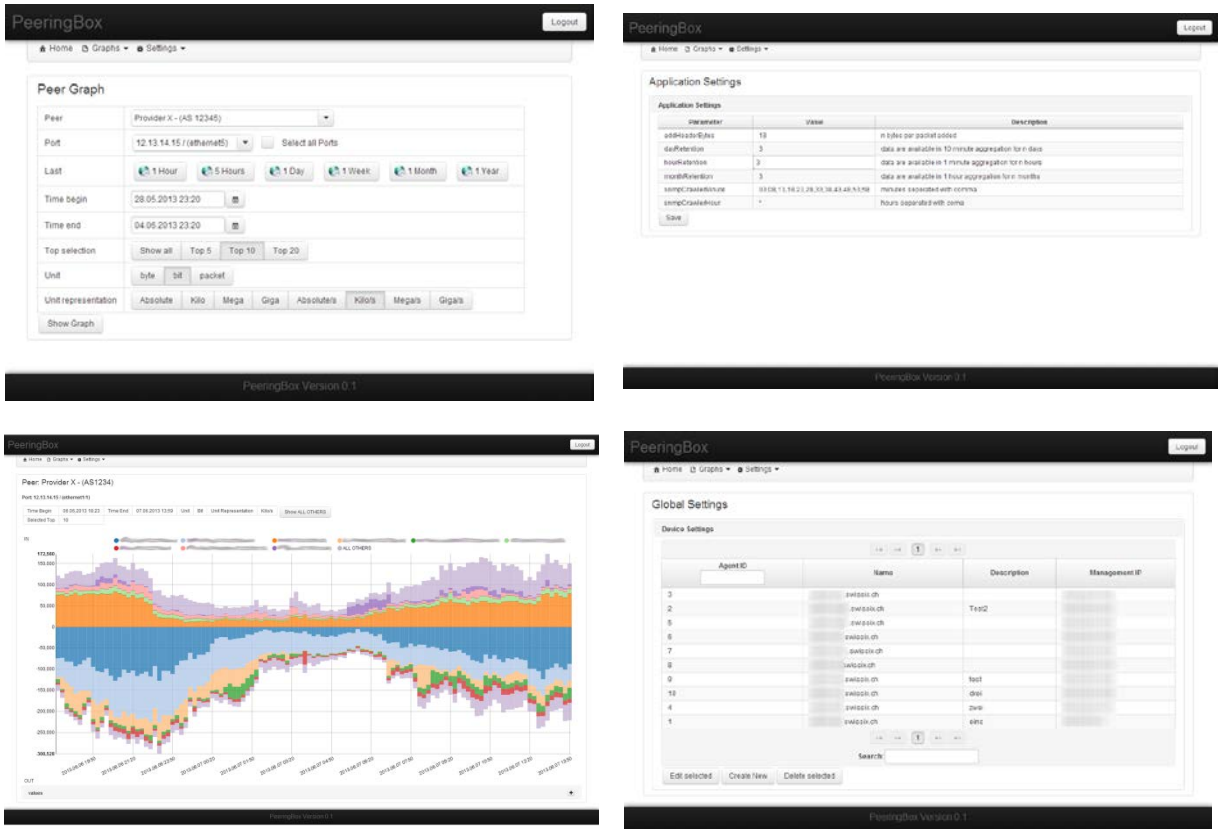


Abbildung 28: Frontend Eindruck

5.4 Datensammlung

5.4.1 SNMP

Das Simple Network Management Protocol (SNMP) ermöglicht, wie es der Name schon sagt, eine einfache Abfrage von Zuständen eines Gerätes über das Netzwerk. Aktuelle Netzwerkgeräte, wie auch die im SwissIX eingesetzten Brocade-Switches, bieten die Kommunikation über dieses Protokoll ebenfalls an. Im Normalfall findet die Kommunikation über UDP/161 statt. Dies ist aber variabel und lässt sich bequem in der Anwendung für jedes einzelne erfasste Netzwerkgerät spezifizieren.

Über SNMP wird vorwiegend auf die folgenden drei Informationen zugegriffen.

Bezeichnung	Fachjargon	OID
Interface description	ifDesc	1.3.6.1.2.1.31.1.1.1.1
Interface alias	ifAlias	1.3.6.1.2.1.31.1.1.1.18
Mac-Adress-Table des Switches	dot1dTPFdbAddress	1.3.6.1.2.1.17.4.3.1.1
	dot1dTPFdbPort	1.3.6.1.2.1.17.4.3.1.2

Diese Object Identifier (OID) bezeichnen jeweils den Weg zu der gewünschten Ressource. SNMP ist wie ein Baum aufgebaut. Je länger die Nummer ist, desto dünner wird der Ast, bis man schlussendlich zum Blatt gelangt, in welchem die gewünschten Informationen enthalten sind. Die verwendeten Abfragen stammen aus den unten aufgeführten Management Information Bases (MIBs), welche standardisiert sind. Der Vollständigkeit halber sind hier beide aufgeführt:

- OID-Tree 1.3.6.1.2.1.17
  - Bezeichnung: Bridge-MIB / RFC 4188
  - http://tools.ietf.org/html/rfc4188
  - http://www.oidview.com/mibs/0/BRIDGE-MIB.html
  - http://www.brocade.com/downloads/documents/html_product_manuals/NOS_MIB_301/wwhelp/wwhimpl/js/html/wwhelp.htm#href=7_bridge-mib.12.3.html
- OID-Tree 1.3.6.1.2.1.31
  - Bezeichnung: If-MIB / RFC 2863
  - http://tools.ietf.org/html/rfc2863
  - http://www.oidview.com/mibs/0/IF-MIB.html
  - http://www.brocade.com/downloads/documents/html_product_manuals/B6910_MIB_2200/wwhelp/wwhimpl/js/html/wwhelp.htm#href=standard_mib.04.15.html

Mit einem SNMP-WALK Befehl kann ein ganzer Unterbaum (Subtree) abgefragt werden. Man erhält eine Liste von Antworten, wobei jeweils nur die Blätter des Baumes angezeigt werden.

Es sind auch graphische Applikationen vorhanden, die solche WALK Befehle ausführen können, doch hier wird lediglich die Verwendung unter Linux gezeigt. Nach der Installation des entsprechenden Pakets mit „apt-get install snmpd“ kann der folgende Befehl ausgeführt werden.

- snmpwalk -v 2c -c community 11.22.33.44 1.3.6.1.2.1.2.2.1.10
  - –v 2c SNMP-Version, die verwendet werden soll
  - –c community Community-String, womit Zugriffsrechte geregelt werden
  - 11.22.33.44 IP-Adresse des Zielservers (Switch)
  - 1.3.6.1.2.1.2.2.1.10 OID, dessen Blätter angezeigt werden sollen.

Im konkreten Falle eines Brocade Switches der SwissIX erhält man dabei folgende fiktiven Antworten:

Bezeichnung	Antwortmöglichkeiten
ifDesc	OID.**135** = STRING ethernet3/12
ifAlias	OID.**135** = STRING Provider X (AS12345, 11.12.13.14, 2001:123:1::b)
dot1dTPFdbAddress	OID.*0.38.11.220.238.128* = STRING: 00 11 22 AA BB CC
dot1dTPFdbPort	OID.*0.38.11.220.238.128* = INTEGER: **135**

Die erhaltenen Daten können alle einem Port dieses Switches zugeordnet werden. Der Port mit der Nummer **135** enthält also die folgenden Informationen:

- Interface Description: ethernet3/12
- Interface Alias: Provider X (AS12345, 11.12.13.14, 2001:123:1::b)
- Mac-Adresse des Peers: 00 11 22 AA BB CC (Zuordnung via *OID*)

5.4.1.1 SNMP4J

Diese Informationen werden von der Applikation interpretiert und verwendet, um Ports, Portallocations, Mac-Adressen und Peers zu erfassen oder zu bestätigen. In der Java-Welt hat sich die Library SNMP4J[7] behauptet. Damit lassen sich einfach und zuverlässig SNMP-Walk Befehle ausführen und eine Liste aller Inhalte der Blätter zurück erhalten.

5.4.1.2 Zeitplan

Die Komponente, welche periodisch abläuft und die Daten der verschiedenen Switches abholt, wurde SNMP-Crawler getauft. Per Default wird diese Klasse alle 5 Minuten gestartet, damit sie ihren Dienst verrichten kann.

Da andere Applikationen wie Observium ebenfalls alle 5 Minuten beginnend bei 00:00 per SNMP Abfragen starten, beginnt die PeeringBox jeweils um 00:03. Dies garantiert, dass sich die breit eingesetzte Applikation Observium nicht mit unserer Applikation überschneidet, denn eine Abfrage dauert durchschnittlich nur gerade 30 Sekunden. Im Weiteren ist diese Einstellung über das Interface konfigurierbar.

[7] http://www.snmp4j.org/

5.4.1.3 Algorithmus

Die Informationen werden nun vom Programm gefunden, jedoch ist dies noch kein Mehrwert, solange die Informationen nicht richtig interpretiert und in der Datenbank persistiert wurden. Zu diesem Zweck wurde ein Algorithmus entwickelt, der gemäss oben definiertem Zeitplan abläuft.

Die unten stehende Graphik zeigt den genauen Ablauf des Algorithmus:

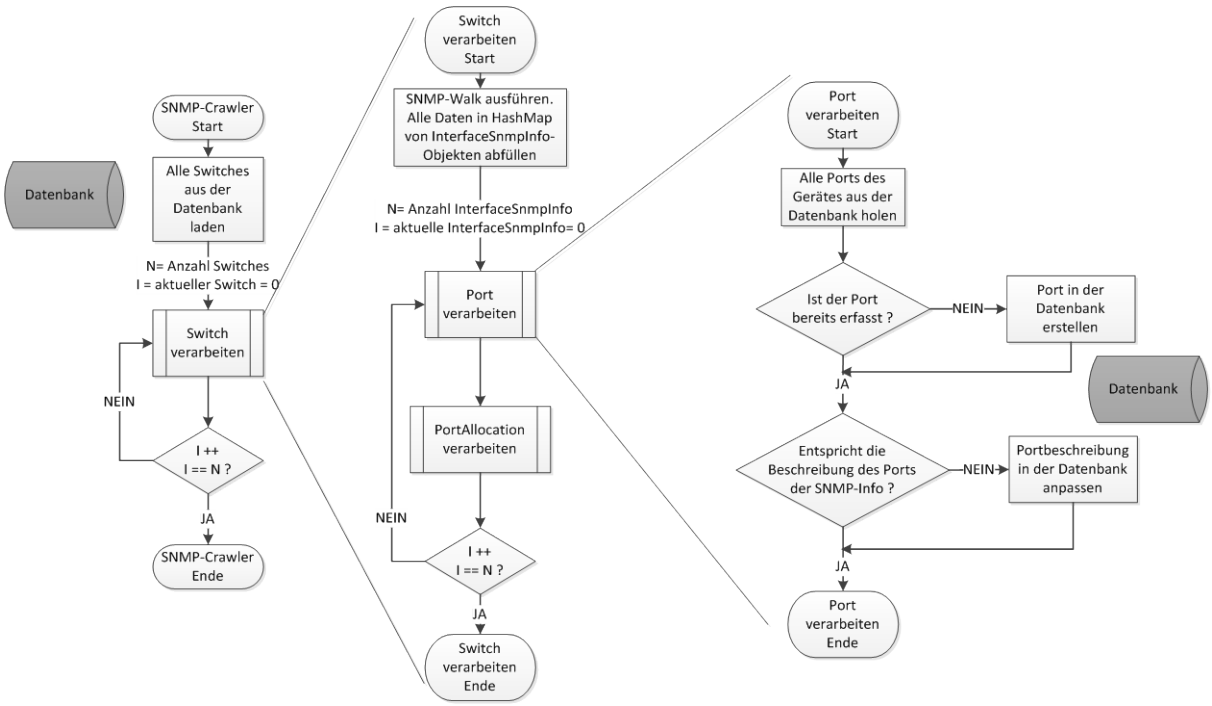


Abbildung 29: SNMP-Crawler Gesamtübersicht Algorithmus

Pro Switch, der verarbeitet wird, werden die per SNMP abgeholten Daten in folgenden Container verpackt. Ein solches Objekt enthält alle Informationen, die benötigt werden.

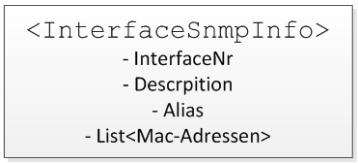


Abbildung 30: SNMP-Crawler Datenstruktur

Im Beispiel von oben würde dies also wie folgt aussehen:

- InterfaceNr: 135
- Description: ethernet3/12
- Alias: Provider X (AS12345, 11.12.13.14, 2001:123:1::b)
- Mac-Adresse des Peers: 00 11 22 AA BB CC

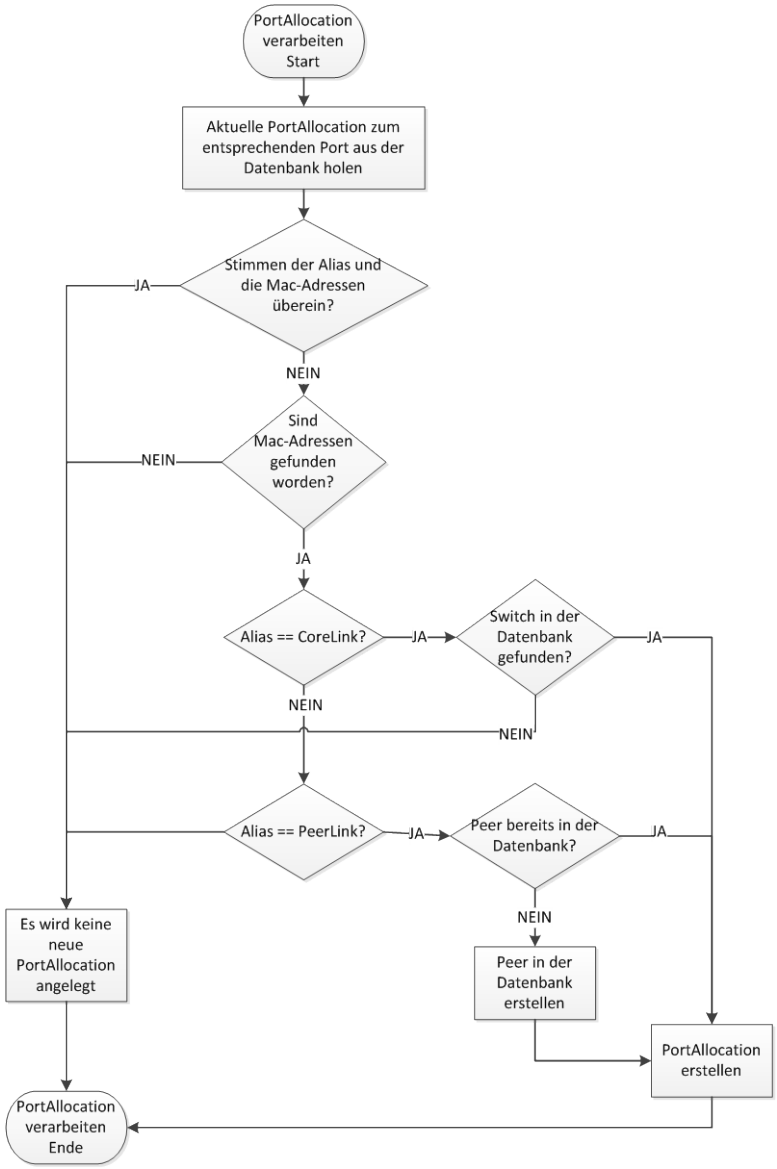


Abbildung 31: SNMP-Crawler Algorithmus Portallocations

Damit entschieden werden kann, ob ein Port nun als CoreLink, oder als PeerLInk verwendet wird, muss der Alias einem bestimmten Muster entsprechen. In Absprache mit unserem Auftraggeber SwissIX gibt es die folgenden Möglichkeiten eines korrekten Alias:

CoreLink	Core: switch-czh (e via G&C)
	Core: switch-czh
	Core: switch-czh.swissix.ch
	Core: DWDM to switch-zh4 C31
PeerLink	Qube ABC ISP (AS1234, 11.12.13.14, 2001:123:1::b)
	Qube ABC ISP (AS1234, 11.12.13.14)
	Qube ABC ISP (AS1234, 2001:123:1::b)

Der Algorithmus kann all diese verschiedenen Alias verarbeiten. Falls ein Alias jedoch keinem der hier festgehaltenen Vorgaben entspricht, wird keine Portallocation erstellt.

5.4.2 sFlow

sFlow ist ein Standard zur Flussüberwachung von Rechnernetzen. Dabei werden die Daten, welche über ein Netzwerkgerät fliessen, gesampelt um die Datenmenge zu reduzieren. Ein “Flow“ (Kombination aus Mac-Source, Mac-Destination, L2-Protokoll, L3-Protokoll) kann jeweils zusammengefasst werden, ohne dass Daten enorm an Aussagequalität verlieren. Zumindest im Rahmen der geforderten Auswertungen.

5.4.2.1 Flows

Bei aktuell 140 Peering-Teilnehmern kann es lediglich $\frac{n*(n-1)}{2} = \frac{140*139}{2} = 9730$ Gesprächspartner geben. Diese sprechen lediglich zwei Sprachen auf Layer 2 (IPv4 / IPv6) und realistisch gesehen lediglich zwei auf Layer 3 (TCP / UDP). Dies entspricht also maximal $9730 * 2 * 2 = 38'920$ Einträgen pro Zeitslot, falls nach den gegebenen Primitiven aggregiert wird. Falls die Daten linear historisiert werden könnten (das heisst, falls pro Sekunde jeder Flow einmal existiert) ergeben sich folgende Werte daraus.

Zeitspanne		Einteilung	Flows
Eine Stunde	60	(Minute / Stunde)	2‘335‘200
Ein Tag	48	(10 Minuten / Tag)	5'604'480
Ein Monat	30	(Stunden / Monat)	28'022'400
Ein Jahr	52	(Tage / Jahr)	13'855'520

5.4.2.2 Evaluierung

Um sFlow Daten in PeeringBox verfügbar zu machen wird ein Tool benötigt, welches die sFlow Daten persistieren kann. Die Anforderungen an so ein Tool sind:

- Daten können in einer Datenbank persistiert werden (vorzugsweise PostgreSQL).
- Daten können den Anforderungen entsprechend Aggregiert werden.
- Daten können über eine Periode zusammengefasst wirden.
- Es soll nicht zu viele Ressourcen benötigen.
- Es muss einfach zu konfigurieren sein.

Da die PeeringBox eine Folgearbeit ist wurde dem Projektteam ein gewisser Teil dieser Arbeit schon abgenommen. Der pmacct sFlow Collector wurde im Rahmen der Vorhergehenden Arbeit schon evaluiert. Das Projektteam der Bachelorarbeit hat entsprechend den Anforderungen von PeeringBox das Tool analysiert. Da pmacct den Anforderungen von PeeringBox entspricht wird dieses auch wieder in PeeringBox verwendet. Damit hat das Projektteam, dass es auf Erfahrungen aus der Studienarbeit zurückgreifen kann.

5.4.2.3 PMACCT

Um die sFlow Daten zu historisieren wird pmacct benutzt. Es erlaubt über verschiedene Primitiven zu aggregieren und fasst Anzahl Pakets und Bytes zu zusammen. Das Tool pmacct erfüllt alle Anforderungen, die an den Datencollector gestellt sind. Je nach Aggregation der sFlow Daten resultiert eine grössere oder kleinere Menge von Datensätzen, welche in die PostgreSQL Datenbank geschrieben werden. Darum ist es nötig einen Konsens zwischen Performance und Detailreichtum der historisierten Daten zu finden. Als Rahmenbedingungen liegen folgende Vorgaben vor:

- Das System soll mit einer Datenlast von bis zu 1000 Mbps klar kommen.
- Das System läuft auf einer VM die von SwissIX bereitgestellt wird.

Pmacct	http://www.pmacct.net/ http://wiki.pmacct.net/

5.4.2.3.1 Daten

Die Daten werden von pmacct in die Datenbank geschrieben. Dabei werden die sFlow Samples nach Zeiteinheiten zusammengefasst. Die aktuelle Konfiguration summiert alle sFlow Samples während einer Minute und schreibt die Summe der Pakete und die Summe der Bytes in ein Feld in der Datenbank. Bei diesem Prozess werden die Primitiven von pmacct berücksichtigt. Die Daten liegen in folgendem Format vor:

Feld	Beispielwert	Beschreibung
id	3	Eindeutige Identifikation pro Eintrag
agent_id	4	Pro Switch statisch definiert
iface_in	322	Eingangsinterface
iface_out	68	Ausgangsinterface
etype	800	EtherType des Ethernet-Frames (IPv4 / IPv6 / ARP / ADR / usw.)
mac_src	00:12:f2:9f:21:02	Mac address source / Quell-Adresse des Layer 2
mac_dst	28:c0:da:01:98:7c	Mac address destination / Ziel-Adresse des Layer 2
port_src	0	Quell-Port im Layer 4 (0, falls nicht well-known)
port_dst	22	Ziel-Port im Layer 4 (0, falls nicht well-known)
ip_proto	6	Protocol-Feld des Layer3-Headers (TCP / UDP / usw.)
tcp_flags	24	Das TCP-Flag des Layer4 Headers (SYN / SYN-ACK / ACK / usw.)
packets	610304	Anzahl der Pakete im Zeitabschnitt
bytes	813217792	Anzahl der Bytes aller Pakete im Zeitabschnitt
stamp_inserted	21.04.2013 11:51	
stamp_updated	21.04.2013 11:52	

Die Anwendung pmacct lässt eine manuell definierte Liste von sogenannten „well-known“-Ports zu. Diese Liste ist aktuell in der Datei „ianaports“ festgehalten. Alle Ports, die nicht in der Datei enthalten sind, werden mit dem Wert 0 erfasst. Die aktuelle Liste ist dem Anhang zu entnehmen.

5.4.2.3.2 Richtigkeit

Im Rahmen der Elaboration Phase haben wir einen Prototyp der Datensammlung mit pmacct erstellt um die Richtigkeit der Daten welche gesammelt werden zu validieren.
Die Genauigkeit beträgt bei einer Sampling-Rate von 512 ungefähr 96%. Der Folgende Graph zeigt das Verhältnis von Prozentualer Fehlerrate zu Anzahl gesampelten Paketen pro 1'000'000 Paketen. Je höher die Sampling-Rate, desto mehr Samples werden erstellt.

Bei einer Sampling-Rate von 1000 werden $\frac{1'000'000\ Pakete}{1'000\ (Samplingrate)} = 1'000$ Samples erstellt werden, welche auf dem Graphen etwa 6% Error entsprechen.

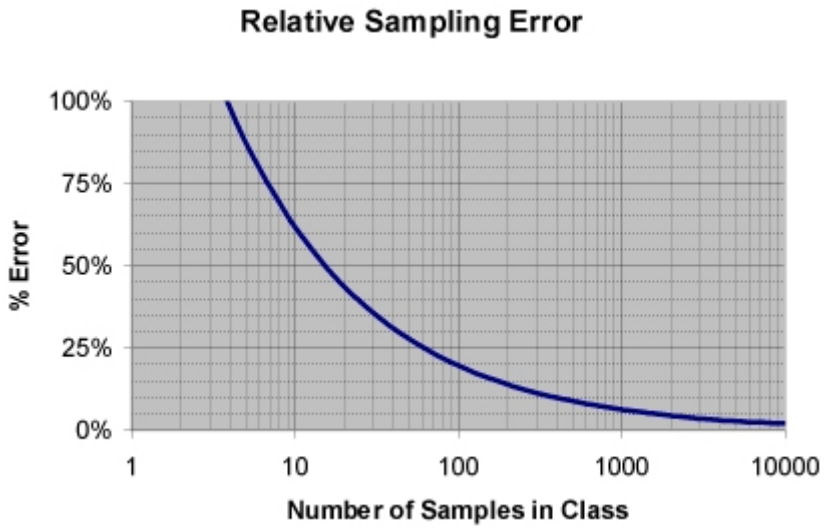


Abbildung 32: sFlow Richtigkeit[8]

Das Verhalten der Genauigkeit im Verhältnis zur Samplingrate wird auf der Webseite von sFlow[9] genau erläutert. Die Validation der gesammelten Daten wird im Abschnitt Machbarkeitsanalyse – Auswertung erläutert.

[8] Quelle: http://www.sflow.org/packetSamplingBasics/index.htm
[9] http://www.sflow.org/packetSamplingBasics/index.htm

5.4.2.3.3 Funktionsweise

Pmacct ermöglicht es, sFlow Streams zu aggregieren und diese über ein Plugin persistent zu halten. Für jeden sFlow Stream ist ein separater pmacct Prozess notwendig, welcher den Stream entgegen nimmt. Folgende Abbildung soll den Prozessablauf von sFlow Daten in die Datenbank verdeutlichen.

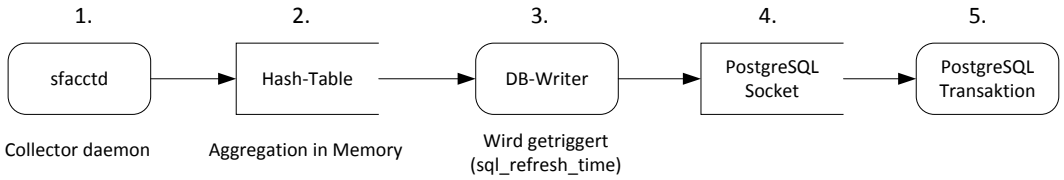


Abbildung 33: pmacct Funktionsweise

1. Der pmacct daemon für sFlow Daten (sfacctd) nimmt sFlow Daten auf einem Port entgegen. Diese Daten werden je nach Konfiguration nach den entsprechenden Primitiven aggregiert. Jeder Flow kann mehrere Plugins besitzen, welch eine eigenständige Hash-Table besitzen.
2. Intern speichert der pmacct Daemon „sfacctd“ die Daten in einer Hash-Table, welche den Speicher im Memory (RAM) darstellt.
3. Der DB-Writer schreibt die Daten aus der Hash-Table zyklisch (über sql_refresh_time konfigurierbar) auf einen Datenbank-Socket.
4. Der Datenbank Socket nimmt die INSERT-Statements des DB-Writer entgegen.
5. Die Datenbank startet eine Transaktion, welche die Daten, welche vom DB-Writer auf den Socket geschrieben wurden, in die Datenbank schreibt.

5.4.2.3.4 Aggregationen

Mit pmacct ist es möglich, die gesammelten Daten zu aggregieren. Dies ist sinnvoll, da die Daten so schon vor dem Speichern in die Datenbank zusammengefasst werden können.
Die Daten werden aggregiert sobald sich die konfigurierten Primitiven nicht unterscheiden. Als Beispiel nehmen wir an, es sind die Primitiven „mac_src“ und „mac_dst“ konfiguriert.

mac_src	mac_dst	packets	bytes
00:16:4d:73:b2:23	00:1f:12:8a:f4:83	512	20480
00:16:4d:73:b2:23	00:1f:12:8a:f4:83	512	752640
00:21:59:0c:bc:84	74:8e:f8:29:08:31	512	20480

Durch die gleiche mac_src und mac_dst der ersten zwei Flows werden diese aggregiert und über die packets und bytes wird die Summe gezogen.

mac_src	mac_dst	packets	bytes
00:16:4d:73:b2:23	00:1f:12:8a:f4:83	1024	773120
00:21:59:0c:bc:84	74:8e:f8:29:08:31	512	20480

5.4.2.3.4.1 Primitiven

Über die primitiven kann gesteuert werden, wie die Daten von pmacct aggregiert werden sollen. Hier eine vollständige Liste der für dieses Projekt interessanten Primitiven.

- tag: Dies stellt ein Device (zum Beispiel ein Switch) dar und wird in der pmacct-config pro Device gesetzt.
- in_iface/out_iface: Das Interface auf einem Device (ein-/ausgehender Verkehr)
- etype: Der EtherType des Ethernet-Frames - L3 Protokoll (IPv4 / IPv6 / ARP / ADR / usw.)
- proto: Das Protocol-Feld des Layer3-Headers – Layer 4 Protokoll (TCP / UDP / usw.)
- src_port/dst_port: Der verwendete Layer4 Port eines Flows (identifiziert die Applikation)
- src_mac/dst_mac: MAC source/destination
- src_host/dst_host: Layer 3 Adresse (ipv4 oder ipv6)
- tcpflags: Bei einer TCP Verbindung werden die TCP Flags gespeichert.

5.4.2.3.5 Machbarkeitsanalyse - Collector

5.4.2.3.5.1 Vergleich

Um die verschiedenen Konfigurationen zu vergleichen, haben wir folgende Parameter analysiert.

- await: durchschnittliche Antwortzeit in ms von I/O-Requests
- %util: Prozent der CPU time welche von einem I/O-Request von der Festplatte benötig wurde

Dieser Vergleich wurde nötig, da wir anfangs Schwierigkeiten mit der Performance der Datensammlung hatten. Durch die Analyse wurde gezeigt, dass der grösste Faktor für die Performance die Festplatte (I/O-Performance) des Servers darstellt. Die Messungen wurden mit den Tools iostat (I/0-Statistiken) und vnstat (Netzwerklast) gemacht.

Konfiguration	Last zur Zeit der Messung	await	%util
Standard	25 Mbps	57.5 ms	3.5%
Port	23 Mbps	69.3 ms	5.8%
Host & Port	22 Mbps	814.8 ms	52.5%

Dies sind die durchschnittlichen Messwerte gemessen über eine halbe Stunde. Die Standard Konfiguration bezieht sich auf die Aggregation mit tag, in_iface, out_iface, etype, src_mac, dst_mac, proto und tcpflags. Für die Konfiguration „Port“ kommen jeweils zusätzlich zur Standardkonfiguration noch src_port, dst_port hinzu. Host & Port erweitert die Port-Konfiguration entsprechend um src_host und dst_host.

Die Historisierung mit pmacct ist mit der aktuellen Hardware nicht lösbar, da jedes Plugin 300Mbyte RAM benötigt. Unsere ursprünglich angestrebte Historisierungs-Strategie (Host&Port) benötigt 4 Plugins pro Netzwerkgerät, womit 12Gbyte RAM erforderlich wären. Auch die %util skaliert schlecht, da jedes Plugin seinen eigenen DB-Writer besitzt (also Faktor 4). Im Test tauchen diese Werte darum nicht auf, da der Memory überläuft und dadurch das System instabil wird.

Der grosse Unterschied zwischen Port und Port & Host entsteht durch die grosse Verteilung der Hosts im Vergleich zu den Ports. Folgende Diagramme beschreiben die Prozentuale Mengenverteilung der Ports und der Hosts während einer Stunde.

Mengenverteilung nach Port

Die Verteilung nach Ports zeigt, dass 80% der Flows durch ungefähr 20 Ports abgedeckt sind. Die Aggregation nach Ports würde die Datenmenge erhöhen aber wäre im Rahmen des Machbaren.

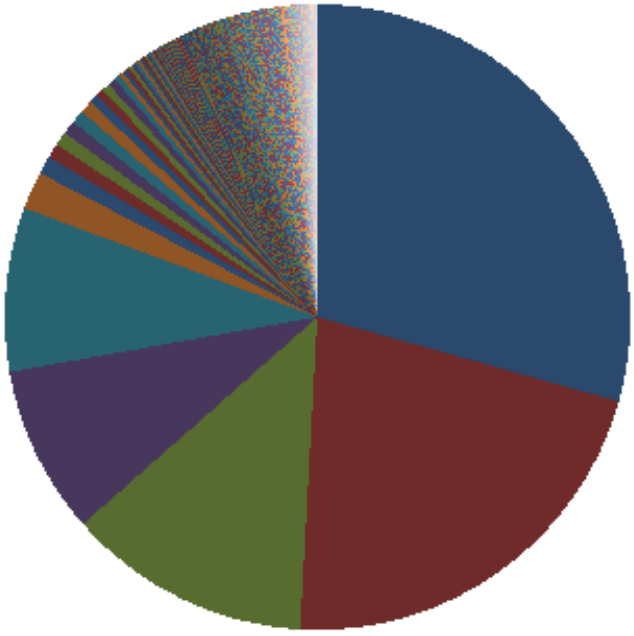


Abbildung 34: Mengenverteilung nach Port

Mengenverteilung nach Host

Die Verteilung nach Hosts zeigt, ganz deutlich, dass eine grosse Anzahl an verschiedenen Hosts im System vorhanden ist. Dadurch entstehen so viele Flows, dass kaum noch Daten aggregiert werden können, was in einer riesigen Datenmenge resultiert.

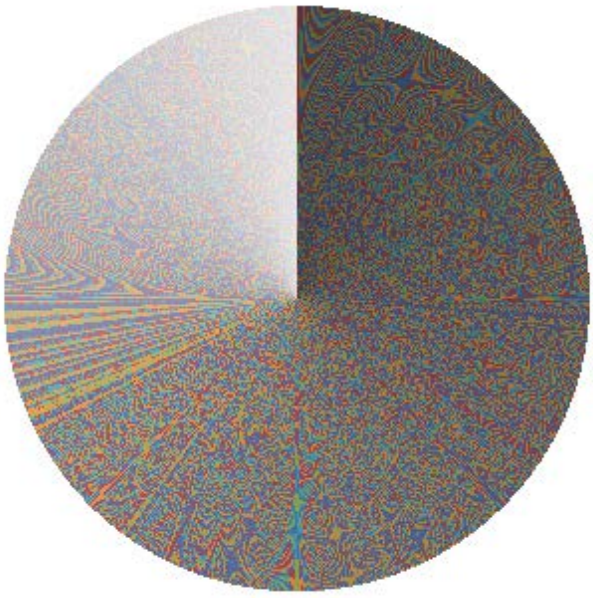


Abbildung 35: Mengenverteilung nach Host

5.4.2.3.5.2 Hochrechnung

Die gegebene Anforderung an die Software besagt, dass 1000 Mbps an Daten verarbeitet werden können. Bei einer linearen Hochrechnung kann dies mit der aktuellen Hardware nicht erreicht werden. Die Annahme, dass sich das Datenaufkommen linear verhält, kann aber nur bei der Konfiguration mit Host getroffen werden, da die Ports sehr ungleichmässig verteilt sind. Das Datenaufkommen bei der Standard und der Port Konfiguration wird viel kleiner ausfallen.

100 Mbps (Faktor 4)

Konfiguration	await	%util
Standard	230 ms	14%
Port	277 ms	23.2%
Host & Port	3‘259.2 ms	210%

1000Mbps (Faktor 40)

Konfiguration	await	%util
Standard	2‘300 ms	140%
Port	2‘772 ms	232%
Host & Port	32‘592 ms	2‘100%

5.4.2.3.5.3 Fazit

Eine Aggregation mit Host ist performancetechnisch nicht anzustreben, da es sehr schlecht skaliert. Schon mit der aktuellen Hardwarekonfiguration kommt das System an seine Grenzen. Die Ressourcen, welche für Auswertungen und für die Historisierung benötigt werden, sind in diesem Konzept noch nicht einberechnet. Unsere Entscheidung fällt darum auf eine Konfiguration, welche neben den Standard-Primitiven maximal die Ports zusätzlich enthält.

5.4.2.3.6 Machbarkeitsanalyse – Auswertung

Um herauszufinden, wie sich verschiedene Konfigurationen bei der Auswertung verhalten, wurden Messungen vorgenommen. Diese Messungen und die Erkenntnisse daraus sind hier fest gehalten.

Die detaillierten Messresultate sind dem Anhang **Fehler! Verweisquelle konnte nicht gefunden werden.** zu entnehmen.

5.4.2.3.6.1 Testbedingungen

Um verschiedene Konfigurationen miteinander vergleichen zu können, mussten vergleichbare Bedingungen geschaffen werden.

Verglichen wurden jeweils Stunden- und Tagestabellen.

- hour Wochentags, jeweils 11:00 bis 12:00 Uhr
- day Wochentags, jeweils einen gesamten Tag

Queries wurden im Voraus definiert, damit bei all den verschiedenen Konfigurationen dieselbe Auswertung gemessen werden konnte. Queries wurden jeweils dann ausgeführt, wenn keine weiteren Arbeiten am System anstanden, die das Resultat verfälschen könnten.

Ports und Mac-Adressen, welche verwendet wurden, sind im Vorfeld eruiert worden. Dazu wurden verschiedene Abfragen gestartet um Versuchswerte zu finden, welche ein durchschnittliches Verhalten an den Tag legt.

Es sind auch mit diesen Massnahmen nur Näherungswerte bestimmbar. Falls zum Messzeitpunkt ein abnormales Verkehrsaufkommen zu verzeichnen war, wirkt sich dies auch auf die gesammelten Daten aus. Bei der Stundenmessung wirkt sich dies ziemlich stark aus. Über einen gesamten Tag kann dies jedoch vernachlässigt werden.

5.4.2.3.6.2 Queries

Die folgenden Queries wurden verwendet um die Abfragezeit zu messen. Diese Queries werden in leicht abgewandelter Form auch von der Applikation verwendet, womit der Realitätsbezug gegeben ist.

Bezeichnung	Query
Query 1 **Abfrage nach:** **- Interface** **Antwort:** **- Interface - Flows**	`SELECT iface_in, iface_out, SUM(packets), SUM(bytes), stamp_inserted FROM z_day_2013_xx_yy WHERE agent_id = 1 AND (iface_in = 199 OR iface_out=199) GROUP BY iface_in, iface_out, stamp_inserted;`
Query 2 **Abfrage nach:** **- Mac-Adresse** **Antwort:** **- Mac-Adresse-Flows**	`SELECT mac_src, mac_dst, SUM(packets), SUM(bytes), stamp_inserted FROM z_day_2013_xx_yy WHERE agent_id = 1 AND (mac_src = '00:1c:0f:5f:80:00' OR mac_dst = '00:1c:0f:5f:80:00') GROUP BY mac_src, mac_dst, stamp_inserted;`
Query 3 **Abfrage nach:** **- Mac-Adresse** **Antwort:** **- verwendete Ports**	`SELECT port_src, port_dst, SUM(packets) FROM z_day_2013_xx_yy WHERE agent_id = 1 AND (mac_src = '00:1c:0f:5f:80:00' OR mac_dst = '00:1c:0f:5f:80:00') GROUP BY port_src, port_dst, stamp_inserted;`

5.4.2.3.6.3 Ergebnisse

Zur Erinnerung sind die verschiedenen Aggregationen noch einmal aufgeführt worden. Der Wert in Klammern zeigt jeweils, wie viele Ports von der Applikation beachtet wurden. Je höher der Wert, desto mehr Einträge resultieren in der Datenbank.

Aggregation 1	Aggregation 2	Aggregation 3	Aggregation 4	Aggregation 5	Aggregation 6
tag, in_iface, out_iface, etype, src_mac, dst_mac, src_port, (5627) dst_port, (5627) proto, tcpflags	tag, in_iface, out_iface, etype, src_mac, dst_mac, src_port, (694) dst_port, (694) proto, tcpflags	tag, in_iface, out_iface, etype, src_mac, dst_mac, src_port, (45) dst_port, (45) proto, tcpflags	tag, in_iface, out_iface, etype, src_mac, dst_mac, src_port, (15) dst_port, (15) proto, tcpflags	tag, in_iface, out_iface, etype, src_mac, dst_mac, src_port, (45) dst_port, (45) proto, -	tag, in_iface, out_iface, etype, src_mac, dst_mac, - - - -

Die Stundenauswertung hängt, wie schon erwähnt, stark vom auftretenden Traffic ab. Der Tagesauswertung ist also mehr Wert zu schenken.

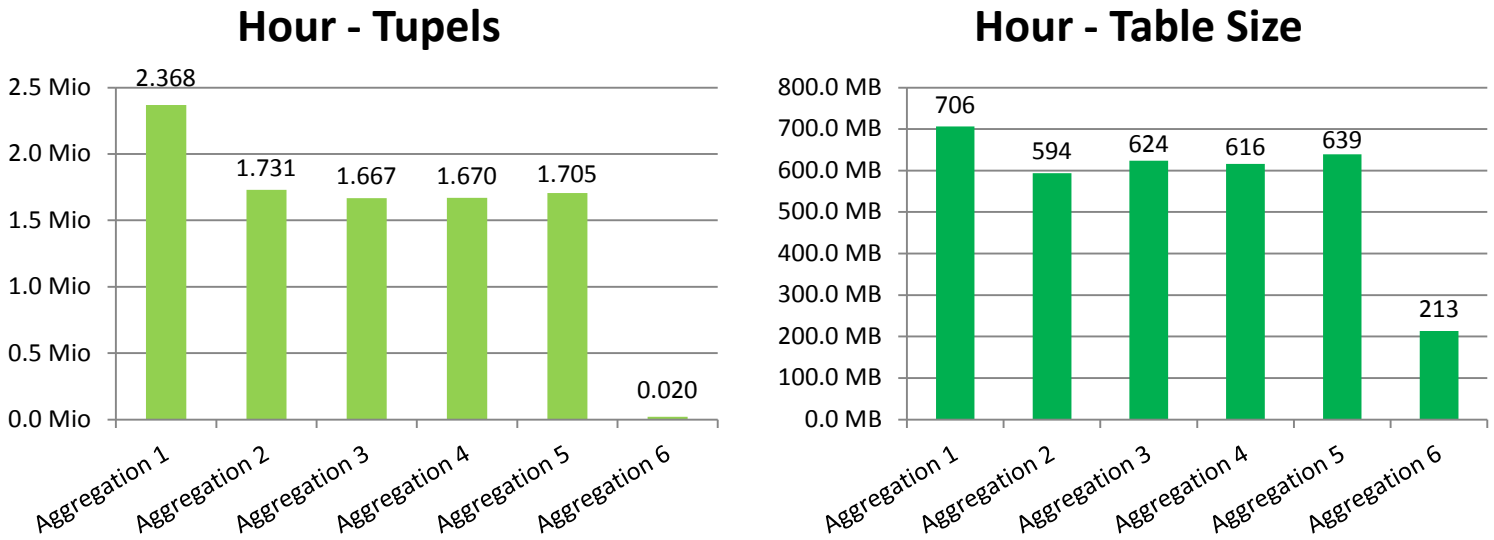


Abbildung 36: Aggregationsvergleich Stunde

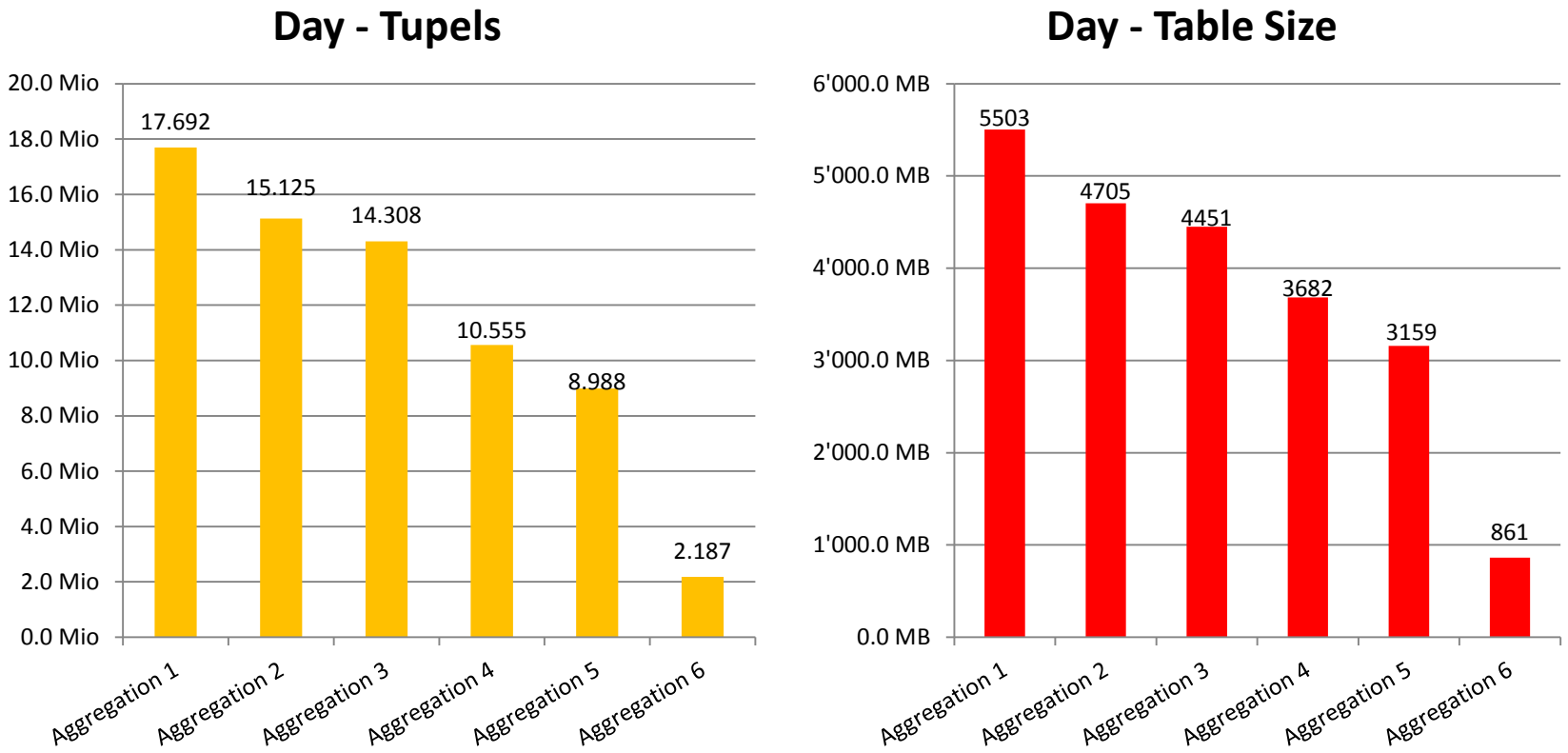


Abbildung 37: Aggregationsvergleich Tag

Zusätzlich wurde die Abfragezeit auf Tabellen der verschiedenen Konfigurationen gemessen. Die Graphik zeigt auch hier eine drastische Abnahme der Antwortzeit.

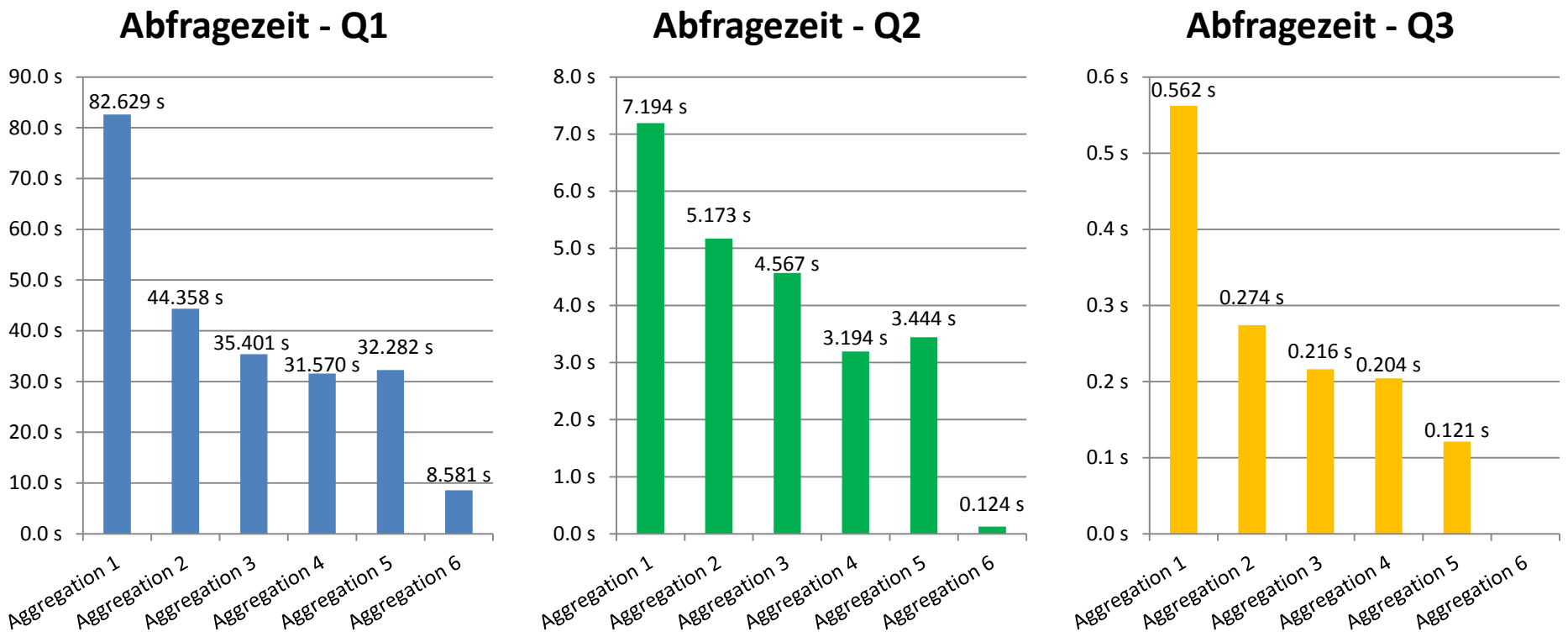


Abbildung 38: Aggregationsvergleich Abfragezeit

5.4.2.3.7 Entscheid

Die Applikation wurde bewusst so flexibel programmiert, dass unterschiedliche pmacct-Konfigurationen keinerlei Anpassungen im Programmcode notwendig machen. Auf Grund der Ergebnisse haben wir uns für die folgenden Aggregationen entschieden.

Aggregation 5	Aggregation 6
tag,	tag,
in_iface,	in_iface,
out_iface,	out_iface,
etype,	etype,
src_mac,	src_mac,
dst_mac,	dst_mac,
src_port, (45)	-
dst_port, (45)	-
proto,	-
-	-

Aggregation 6 - Bachelorarbeit

Mit der im Rahmen der Bachelorarbeit verfügbaren Hardware, waren wir gezwungen nur die absolut notwendigen Attribute für die Aggregation zu verwenden. Dies ermöglicht uns Abfragen, die innert tragbarer Zeit beantwortet wurden. Auswertungen über lange Perioden sind selbst jetzt noch sehr Zeitintensiv.

Aggregation 5 - Produktiveinsatz

Im Produktiveinsatz mit besserer Hardware, empfehlen wir die Aggregation 5. Diese schlisst in Tests gut ab und lässt Auswertungen nach Ports zu. Dabei sind lediglich 45 weit verbreitete Ports involviert. Alle anderen Ports werden „genullt“. Die Tests haben ebenfalls ergeben, dass bei dieser Selektion der Ports ein gutes Mittelmass zwischen Aussagekraft und Ressourcenschonung gefunden werden konnte.

5.4.2.4 Auswertung / Validierung

Bei den gesammelten Daten kann in einem ersten Schritt lediglich mit Menschenverstand abgeschätzt werden, ob die Werte realistisch sind. Dies genügt aber nicht. Da der SwissIX das Überwachungsprogramm „Observium“ einsetzt, wird uns damit eine Möglichkeit geboten, unsere gemessenen Werte zu validieren.

5.4.2.4.1 Vergleich mit Observium

5.4.2.4.1.1 Erläuterungen

Observium bezieht seine Daten über SNMP direkt von den Geräten. Es fragt jeweils den Stand eines Interfaces ab und speichert den relativen Zuwachs an Paketen und geflossenen Bytes in einer Datenbank. Dabei gibt es zwei grundlegende Unterschiede, welche beachtet werden müssen:

Pmacct	Observium
Pmacct summiert den Layer 2 Payload. *(The portion of the packet accounted starts from the IPv4/IPv6 header (inclusive) and ends with the last bit of the packet payload. This means that are excluded from the accounting: packet preamble (if any), link layer headers (e.g. ethernet, llc, etc.), MPLS stack length, VLAN tags size and trailing FCS (if any). This is the main reason of skews reported while comparing pmacct counters to SNMP ones.)*[10]	Es werden über einen Interface-Counter die Anzahl Bits welche über das Interface gehen, gezählt. Dieser Counter wird über SNMP ausgelesen.
Mit sFlow wird gesampelt.	Jedes Bit wird gezählt.

Durch diese Unterschiede entsteht eine Abweichung, welche sich im direkten Vergleich bemerkbar macht. Zusätzlich macht das Accounting mit sFlow nur Sinn, wenn genügend Daten über ein Gerät fliessen, da dadurch die Abweichung durch das Sampling weniger ins Gewicht fällt. Die Headerbytes können von der Anwendung addiert werden. Dazu kann die Anzahl der Bytes im UI festgelegt werden.

Im Weiteren werden per SNMP sowohl die eingehenden, wie auch die ausgehenden Pakete mit gezählt. Das Sampling mit sFlow findet jedoch jeweils nur eingehend statt.

Im folgenden Abschnitt wird ein kompletter Tag, sowie eine komplette Woche miteinander verglichen. Die Abweichungen liegen lediglich im einstelligen Prozentbereich. Dies wurde anhand von Datenbankauswertungen und Excel in Bezug auf Auswertungen von Observium verifiziert. Zur Visuellen Darstellung sind in den Folgeseiten Beispiele visueller Auswertungen zu sehen. Die Observium-Daten liegen in einer Granularität von fünf Minuten vor, wo hingegen die Auswertung der Applikation „PeeringBox“ in einer Granularität von einer Minute gegeben ist. Kurzzeitige Spitzen dürfen nicht zu stark beachtet werden.

[10] http://www.pmacct.net/FAQS-0.14.3

5.4.2.4.1.2 Visueller Vergleich (Tag)

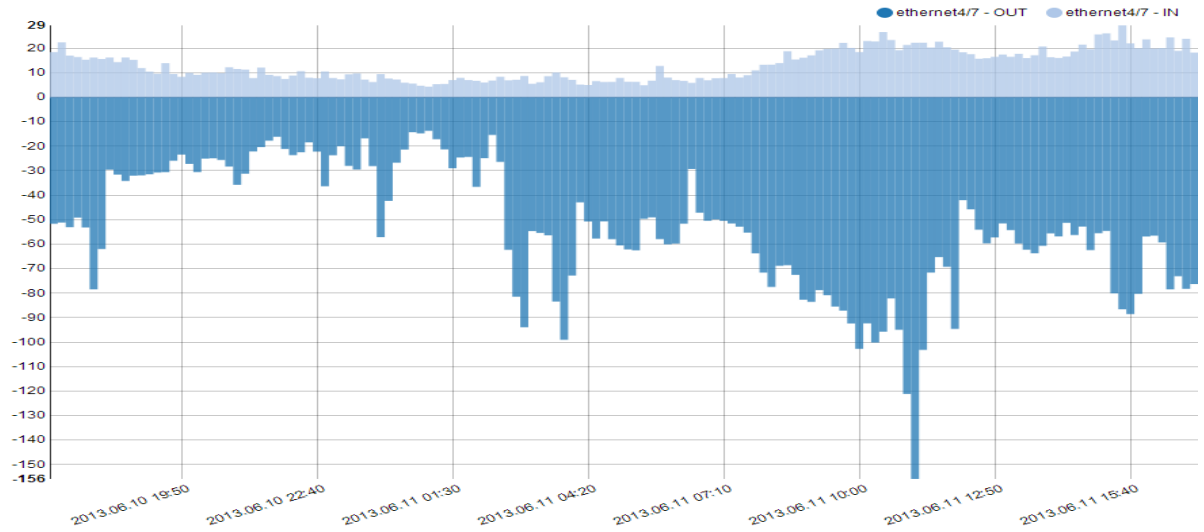


Abbildung 39: Auswertung SFlow-Daten (PeeringBox)

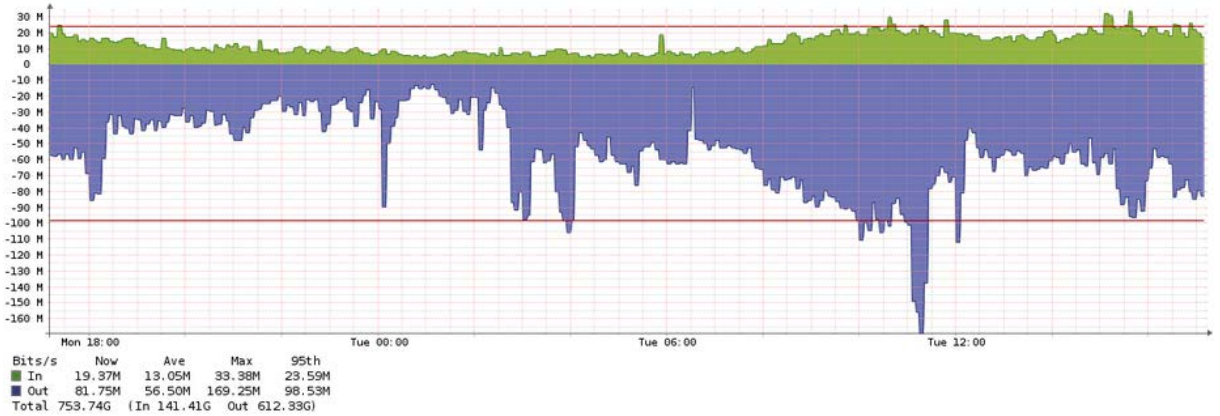


Abbildung 40: Auswertung SNMP-Daten (Observium)

5.4.2.4.1.3 Visueller Vergleich (Woche)

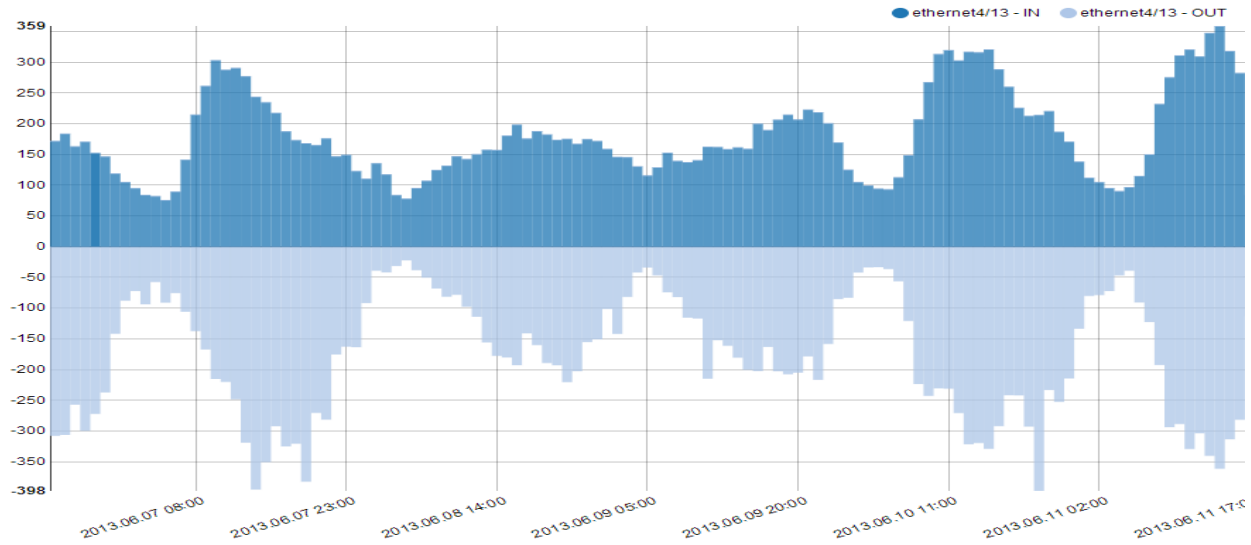


Abbildung 41: Auswertung SFlow-Daten (PeeringBox)

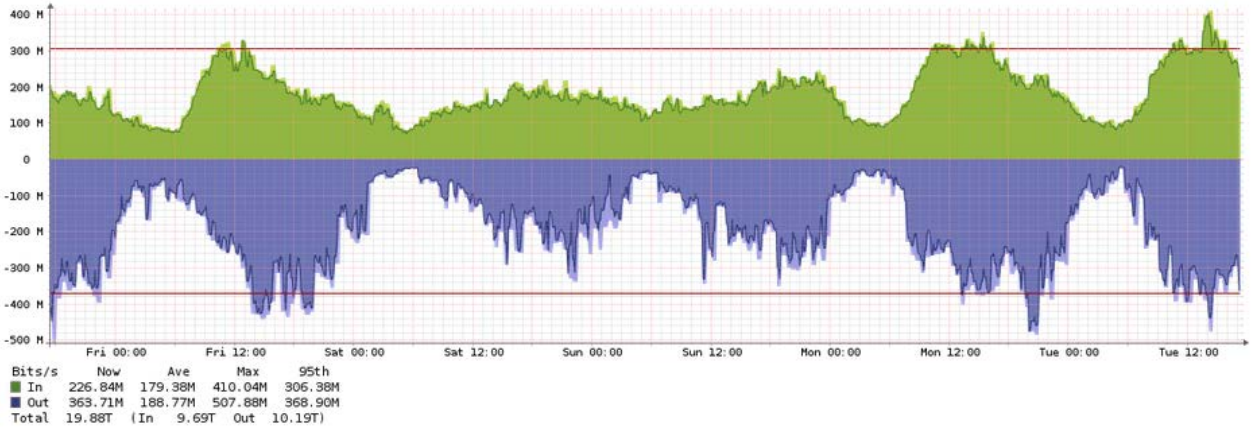


Abbildung 42: Auswertung SNMP-Daten (Observium)

5.5 Historisierung

5.5.1 Funktionsweise

Die gesammelten Daten werden periodisch in eine gröbere Granularität gebracht. Mehrere GlassFish-Timer werden beim Start der Applikation im System registriert. Diese Timer haben einen fixen Ablaufplan. Sobald der Timer abgelaufen ist, wird die Methode mit der entsprechenden Annotation (@Timeout) ausgeführt. Diese Methode instanziiert jeweils den entsprechenden Cleanup-Task und führt darauf die run()-Methode aus. Nach Ablauf wird der Timer darüber informiert, ob der Prozess erfolgreich war oder nicht. Daraufhin „schläft“ der Timer erneut.

Alle Timer erben von der TimerBase-Klasse. Damit wird sichergestellt, dass alle dasselbe Verhalten an den Tag legen. Innerhalb der einzelnen Timer sind lediglich der Ablaufplan und die Aktion, die beim Ablauf des Timers ausgeführt werden soll, definiert. Die verschiedenen CleanupTasks sind ähnlich aufgebaut. Die Ablauflogik ist in der TaskCleanupBase-Klasse definiert. Die einzelnen Tasks implementieren lediglich die notwendigen Queries und Parameter, die den einzelnen Task ausmachen.

Die folgende Graphik zeigt die Programmstruktur der Historisierung.

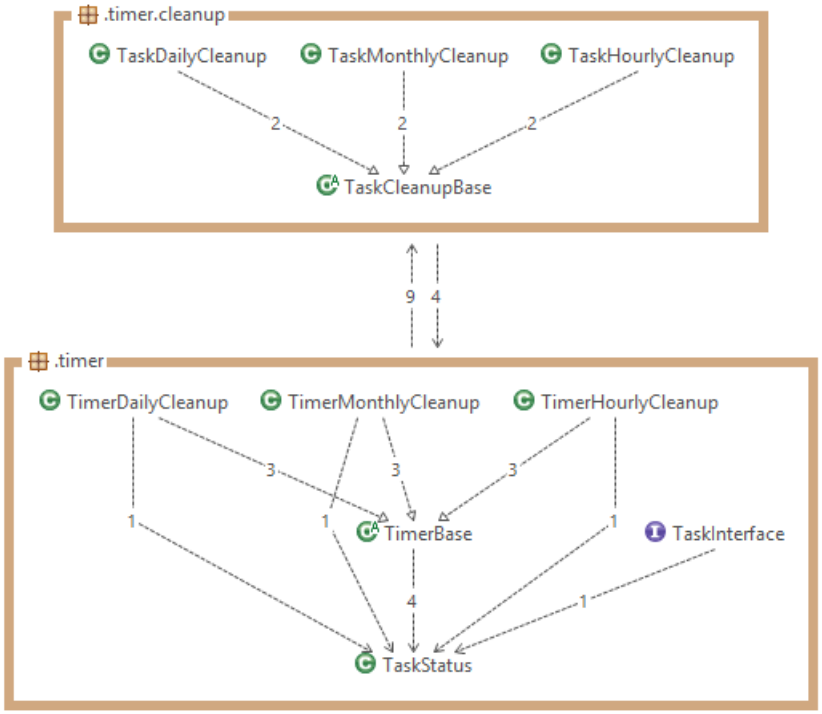


Abbildung 43: Historisierung Programmstruktur

Falls ein neuer Task erstellt werden soll, sollte die Klasse TaskCleanupBase erweitert (extend) werden. Dabei wird man gezwungen (abstract) die folgenden Methoden zu implementieren.

```
// Wird verwendet um die richtigen Tabellen aus dem System zu laden.
protected String getOldPrefix() {return null;}
protected String getNewPrefix() {return null;}
protected String getTableSchemaName() {return null;}

// Check, ob die entsprechende Tabelle bereits verarbeitet werden sollen.
protected int getYearDelta() {return 0;}
protected int getMonthDelta() {return 0;}
protected int getDayDelta() {return 0;}
protected int getHourDelta() {return 0;}

// Falls bei der Historisierung eine PostgreSQL-Funktion verwendet wird, kann dies so
festgelegt werden.
protected boolean isFuncRequired() {return false;}
protected String getFuncName() {return null;}
protected String getCreateFuncQuery() {return null;}

// Diese Strings enthalten die Queries, die in Kombination eine saubere Historisierung
ermöglichen.
protected String getCreateNewTableQuery(String newTableName) {return null;}
protected String getArchiveQuery(String newTableName, String oldTableName) {return null;}
protected String getDropOldTableQuery(String oldTableName) {return null;}
```

Interessant ist dabei der folgende Ablauf:

1. Falls die Historisierung eine neue Tabelle kreieren muss, wird dies mit dem Query der Methode getCreateNewTableQuery() gemacht.
2. Die Daten der Tabelle mit feiner Granularität wird in die neue Tabelle geschrieben. Dies geschieht über das Query, das die Methode getArchiveQuery() liefert. Dabei wird auch festgelegt, wie zusammengefasst werden soll (neue Granularität).
3. Sobald die Historisierung abgeschlossen ist, muss die alte Tabelle gelöscht werden. Diese Query erhält die Anwendung dank der Methode getDropOldTableQuery().

Schritt 2 und 3 müssen dabei zwingend miteinander ausgeführt werden. Nur so kann garantiert werden, dass zu keinem Zeitpunkt Duplikate oder lückenhafte Daten entstehen. Dies wird sichergestellt indem dafür jeweils eine eigenständige Usertransaction erstellt und verwendet wird. Sollte die Query fehlschlagen, so wird keine der beiden Aufgaben persistiert.

5.5.2 Scheduler

5.5.2.1 Zeitplan

Es sind drei Timer definiert, die folgende Ablaufpläne hinterlegt haben. Diese Ablaufpläne werden beim Start der Applikation aus der Datenbank geladen und sind somit variabel und einfach über das UI anpassbar.

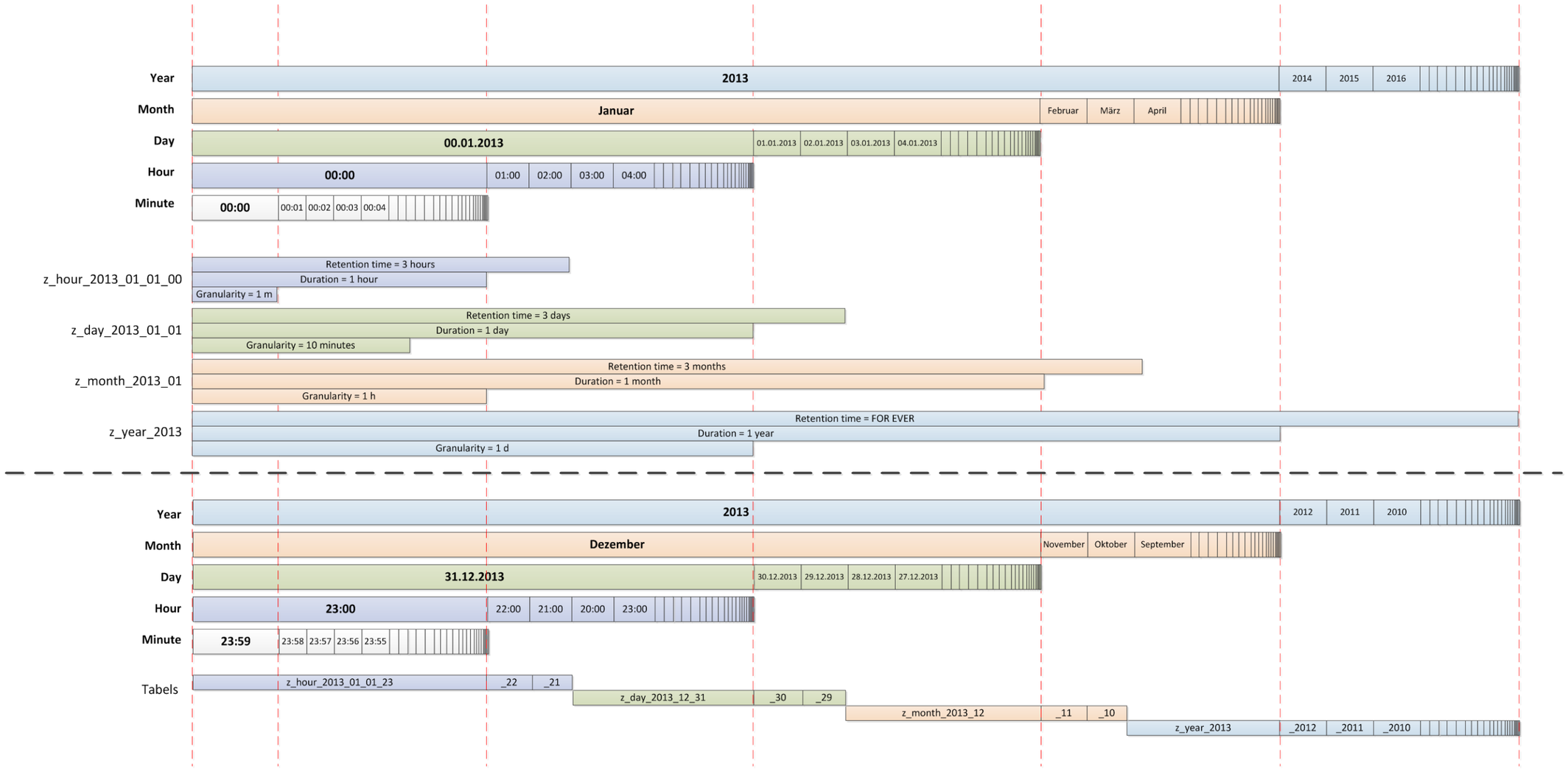
Timer	Ablauf
Hourly Cleanup	Jede Stunde jeweils um xx:10 Uhr
Daily Cleanup	Täglich jeweils um 02:20 Uhr
Monthly Cleanup	Wöchentlich jeweils am Mo um 03:30 Uhr

5.5.2.2 Tasks

Beim Ablauf der Timer werden untenstehende Tasks angestossen. Die Namensgebung bezieht sich jeweils auf die Tabelle, die durch den Task aufgeräumt (cleanup) wird. Die Delta-Zeit wird ebenfalls aus der Datenbank geladen und ist somit bequem über die Applikationseinstellungen definierbar.

Task	Was	Delta	Granularität (von-> nach)
Hourly Cleanup	hour_ nach day_	3 hours	1 min -> 10 min
Daily Cleanup	day_ nach month_	3 days	10 min -> 1 Stunde
Monthly Cleanup	month_ nach year_	3 month	1Stunde -> 1 Tag

5.5.2.3 Gesamtübersicht sFlowDaten



Infos

Allgemein:
Die Zeitskala wurde bewusst exponentiell schrumpfend dargestellt, um mehr Fokus in den feinen Zeiteinheiten zu haben. Auf der linken Seite werden kurze Momente viel breiter dargestellt, als das nach einem Jahr auf der rechten Seite der Fall ist.

Oben:
Jeweils lediglich die Konfiguration der einzelnen Tabellen. Die Graphik zeigt eine Zeitlinie beginnend am 01. Januar 2013 00:00:00 Uhr
- Die «Retention time» beschreibt, wie lange die Tabelle in der Datenbank bestehen bleibt.
- Die «Duration» beschreibt jeweils die Zeit, welche in einer einzelnen Tabelle zu finden ist. Diese Zeit kann auch dem Namen der Tabelle entnommen werden.
- Die «Granularity» beschreibt hierbei wie aggregiert wird. Im Beispiel der «Hour»-Tabelle ist dies eine Minute.

Unten:
Diese Darstellung zeigt eine Momentaufnahme am Ende des Jahres 2013. Also am 31. Dezember 2013 - 23:59:59 Uhr an. Zu erkennen sind die einzelnen Tabellen, welche zu diesem Zeitpunkt im System vorhanden sind. Die hinteren Rechtecke, welche jeweils nur noch eine knappe Beschreibung wie «_22» enthalten, sind wie folgt zu interpretieren: Die hinterste Zahl des Tabellennamens ist jeweils durch die Zahl des entsprechenden Rechtecks zu ersetzen. Somit referenziert die Zahl immer auf die entsprechende Stunde, den entsprechenden Tag, den entsprechenden Monat oder das entsprechende Jahr. Die Tabellen sind bewusst mit dem Prefix «z_» erstellt worden, um Verwechslungen mit anderen Tabellen zu verhindern. Der Buchstabe «z» wurde dabei hauptsächlich gewählt, damit die Tabellen im Datenbanksystem zuunterst aufgeführt werden.

5.5.3 Hochrechnung

5.5.3.1 Berechnungsgrundlage

Die Tabellen „z_hour_“ und „z_day_“ entsprechen jeweils Messwerten. Die Tabellen z_month_ und z_year_ wurden wie folgt errechnet.

5.5.3.1.1 Tabellengrösse

Da ein Flow mehrere Male pro Zeitabschnitt vorkommen kann, wird durch eine Aggregation die Datenmenge kleiner. Dieses Phänomen konnten wir mit einem Multiplikationsfaktor abbilden. Die Aggregation 5 hat durch die Verwendung von Ports und weiteren Protokollinformationen eine sehr viel grössere Anzahl von unterschiedlichen Flows. Beobachtungen zwischen z_hour und z_day haben gezeigt, dass der Faktor 2 einen guten Näherungswert zulässt. Bei der Aggregation 6 hat sich 1.6 als Richtwert ergeben.

$$Aggregation\ 5 \rightarrow\ size_m = size_d * \left(\frac{amount_m}{amount_d}\right) * 2$$

$$Aggregation\ 6 \rightarrow\ size_m = size_d * \left(\frac{amount_m}{amount_d}\right) * 1.6$$

5.5.3.1.2 Komplettes Jahr

Gemäss dem Abschnitt „**Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.**“, sind jeweils mit Ausnahme der Jahrestabelle drei Tabellen jeder Historisierungsstufe vorhanden. Die Summe setzt sich also wie folgt zusammen:

$$size_{Gesamt} = 3 * size_h + 3 * size_d + 3 * size_m + \frac{9}{12} * size_y$$

Für jedes Weitere Jahr kommt jeweils ein $size_y$ dazu.

5.5.3.2 Berechnung

Diese Berechnungen führen zu folgenden Datengrössen.

5.5.3.2.1 Aggregation 6 – Bachelorarbeit

Table	Granularität	Anzahl Zeitslots	Berechnung	Total (MB)
z_hour_	1 min	60	-	213
z_day_	10 min	144	-	861
z_month_	1 h = 60 min	720	861 * 720 / 144 * 1.6	6‘888
z_year_	1 d = 1440 min	365	6888 *365 / 720 * 1.6	5‘586

$$\text{size}_{\text{Gesamt}} = 3 * 213 + 3 * 661 + 3 * 6‘888 + \frac{9}{12} * \ 5‘586 = 27‘475 \text{ MB} \sim 27.5 \text{ GB}$$

Ein gesamtes Jahr an Daten benötigt laut Berechnung lediglich 27.5 Giga Byte. Jedes weitere Jahr belastet den Server um zusätzliche 5.5 Giga Byte.

5.5.3.2.2 Aggregation 5 – Produktiveinsatz

Table	Granularität	Anzahl Zeitslots	Berechnung	Total (MB)
z_hour_	1 min	60	-	639
z_day_	10 min	144	-	3‘159
z_month_	1 h = 60 min	720	3‘159 * 720 / 144 *2	31‘590
z_year_	1 d = 1440 min	365	31‘590*365 / 720 * 2	32‘028

$$size_{Gesamt} = 3 * 639 + 3 * 3‘159 + 3 * 31‘590 + \frac{9}{12} * \ 32‘028 = 130‘185 \ MB \sim 130 \ GB$$

Ein komplettes laufendes Jahr benötigt hier voraussichtlich etwa 130 Giga Byte. Jedes weitere verursacht einen zusätzlichen Platzverbrauch von 32 Giga Byte.

5.6 Auswertung

Die Auswertung findet im Wesentlichen in drei Schritten statt. Diese laufen in der gegebenen Reihenfolge ab, sobald der User auf der Seite das Formular abschickt.

1. SQL-Abfrage
    - Die ausgewählten Informationen werden in eine Query verpackt und an den Datenbankserver übermittelt.
2. Datenaufbereitung
    - Die Antwort des Datenbankservers wird in der Anwendung aufbereitet indem Optimierungs-, Sortierungs- und Umrechnungsarbeiten vollzogen werden.
3. Visualisierung
    - Die aufarbeiteten Daten werden in ein JSON-Objekt abgefüllt und dem User präsentiert.

Die einzelnen Schritte werden in den folgenden Abschnitten genauer erläutert.

5.6.1 SQL-Abfrage

Damit Daten ausgewertet werden können, müssen die Daten zuerst aus der Datenbank gelesen werden. Dazu werden SQL-Queries verwendet, die standardisiert sind und somit auch auf anderen Datenbanksystemen zum Einsatz kommen können.

Hier ein Beispiel eines solchen Statements. Dieses zeigt lediglich den Aufbau. In den folgenden Abschnitten sind Queries auf konkrete Fälle angewendet.

```
SELECT    attribute,
          SUM(attribute2)
FROM      tablename
WHERE     attribute3 = condition
      AND attribute4 = condition2
      OR attribute5 = condition3
GROUP BY  attribute,
          attribute6
ORDER BY  attribute,
          attribute6;
```

5.6.1.1 Peering

Dieses Query wird an die Datenbank gesendet, sobald ein User die „Peering“-Auswertung darstellen möchte.

Hierbei wird zusätzlich beachtet, ob in der entsprechenden Zeit eine Portallocation vorhanden war. Es werden lediglich diese Zeitabschnitte ausgewertet, in denen eine gültige Portallocation des Peers besteht. Dies ist eine zwingend notwendige Massnahme, um verhindern zu können, dass Peers Werte anderer Peers einsehen könnten. Falls der Peer im Zeitabschnitt zwei verschiedene Portallocation besitzt, werden beide ausgewertet, was die Komplexität der Query steigert und die Antwortzeit verlängert. Falls keine Portallocation, oder keine Mac-Adresse gefunden wird, wird keine Query an die Datenbank gesandt.

```
SELECT
            t.mac_src,
            t.mac_dst,
            SUM(t.packets)          AS packets,
            SUM(t.bytes)            AS bytes,
            t.stamp_inserted        AS stamp_inserted,
            t.period
FROM        z_base t
WHERE       (t.agent_id = 7
      AND (((t.mac_src = '11:22:33:44:55:66'
        OR t.mac_dst = '11:22:33:44:55:66')
      AND t.stamp_inserted >= '2013-06-09 13:18'
      AND t.stamp_inserted <= '2013-06-09 14:18')))
GROUP BY    t.mac_src,
            t.mac_dst,
            t.period,
            t.stamp_inserted;
```

Das oben stehende Query resultiert in der untenstehenden Antwort.

	mac_src macaddr	mac_dst macaddr	packets numeric	bytes numeric	stamp_inserted timestamp without time zone	period integer
1	00:1f:6c:83:fe:19	00:12:f2:94:ea:04	2048	135168	2013-06-09 13:22:00	1
2	00:12:f2:94:ea:04	00:26:0b:dd:be:c0	40960	6213632	2013-06-09 13:52:00	1
3	00:12:f2:94:ea:04	00:24:38:a4:a9:07	2048	122880	2013-06-09 13:21:00	1
4	00:1f:12:8a:f4:83	00:12:f2:94:ea:04	475136	710156288	2013-06-09 13:43:00	1
5	40:55:39:48:04:3f	00:12:f2:94:ea:04	2048	225280	2013-06-09 14:07:00	1
6	c4:64:13:9f:3e:80	00:12:f2:94:ea:04	2048	184320	2013-06-09 14:01:00	1
7	84:78:ac:42:79:5c	00:12:f2:94:ea:04	2048	124928	2013-06-09 14:04:00	1
8	5c:5e:ab:0d:d3:72	00:12:f2:94:ea:04	12288	878592	2013-06-09 13:22:00	1
9	00:21:59:a2:90:a5	00:12:f2:94:ea:04	18432	13588480	2013-06-09 13:58:00	1
10	00:12:f2:94:ea:04	00:21:59:0c:bc:84	8192	1828864	2013-06-09 13:41:00	1
11	00:12:f2:94:ea:04	00:24:38:a6:f8:63	14336	6717440	2013-06-09 13:57:00	1
12	00:12:f2:94:ea:04	00:25:45:00:0a:c0	4096	270336	2013-06-09 14:01:00	1
13	c4:7d:4f:21:8f:12	00:12:f2:94:ea:04	14336	6594560	2013-06-09 13:32:00	1
14	28:94:0f:fd:49:40	00:12:f2:94:ea:04	20480	18106368	2013-06-09 14:06:00	1
15	00:1f:12:8a:f4:83	00:12:f2:94:ea:04	567296	844472320	2013-06-09 13:36:00	1

Abbildung 44: Query-Resultat Peer

5.6.1.2 Core (Normal & Observium)

Da diese Auswertungen nur den Mitarbeitern des SwissIX zur Verfügung stehen, muss hier nicht geprüft werden, ob Portallocations vorhanden sind. Dies vereinfacht die Abfrage erheblich.

```
SELECT      t.iface_in,
            t.iface_out,
            SUM(t.packets) AS packets,
            SUM(t.bytes)          AS bytes,
            t.stamp_inserted      AS stamp_inserted,
            t.period
FROM        z_base t
WHERE       t.agent_id = 3
      AND (t.iface_in = 4
        OR t.iface_out=4)
      AND t.stamp_inserted >= '2013-06-09 13:18'
      AND t.stamp_inserted <= '2013-06-09 14:18'
GROUP BY    t.iface_in,
            t.iface_out,
            t.period,
            t.stamp_inserted;
```

Das oben stehende Query resultiert in untenstehendem Resultat.

	iface_in bigint	iface_out bigint	packets numeric	bytes numeric	stamp_inserted timestamp without time zone	period integer
1	24	4	167936	73031680	2013-06-09 13:57:00	1
2	4	24	6144	3643392	2013-06-09 14:12:00	1
3	24	4	34816	6344704	2013-06-09 14:17:00	1
4	4	24	18432	12066816	2013-06-09 14:05:00	1
5	4	0	2048	167936	2013-06-09 14:05:00	1
6	4	24	47104	31240192	2013-06-09 13:28:00	1
7	24	4	34816	14856192	2013-06-09 14:07:00	1
8	4	24	12288	18636800	2013-06-09 14:14:00	1
9	24	4	24576	4702208	2013-06-09 14:09:00	1
10	4	24	53248	18305024	2013-06-09 13:44:00	1
11	4	24	8192	616448	2013-06-09 14:07:00	1
12	24	4	188416	94638080	2013-06-09 13:28:00	1
13	4	24	6144	552960	2013-06-09 14:09:00	1
14	24	4	12288	2822144	2013-06-09 14:14:00	1
15	24	4	180224	47310848	2013-06-09 13:44:00	1

Abbildung 45: Query-Resultat Interface

5.6.1.3 Granularität im Graph

Die PeeringBox benutzt verschiedene Granularitäten für die verschiedenen Historisierungsstufen. Damit im Graphen die Daten einheitlich angezeigt werden, müssen sie je nach ausgewähltem Zeitintervall vereinheitlicht werden. Diese Berechnung findet bereits auf der Ebene der Datenbank statt, damit sich die Software nicht mehr darum kümmern muss.

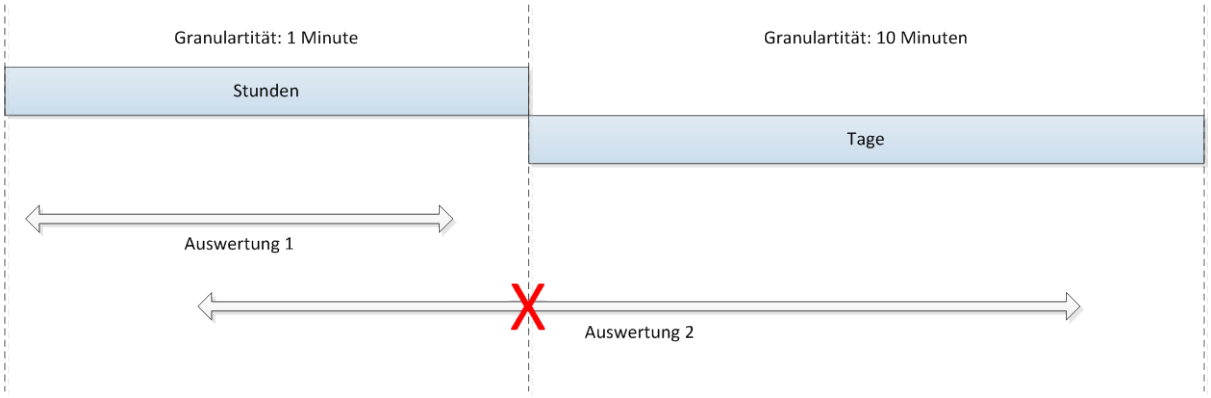


Abbildung 46: Granularität am Grenzübergang

- Auswertung 1: Die Granularität muss nicht geändert werden.
- Auswertung 2: Die Granularität aus den Stundentabellen muss auf die der Tagestabellen hochgerechnet werden.

Immer wenn eine Grenze der Historisierungsstufen überschritten wird, muss jeweils die kleinere Granularität der grösseren angepasst werden. Dabei benutzt die PeeringBox, genau wie bei der Historisierung, die date_trunc() Funktion von PostgreSQL.

5.6.2 Datenaufbereitung

Die erhaltenen Resultate aus dem Datenbanksystem müssen aufbereitet werden, damit die Daten der Auswahl des Users entsprechen.

Dieser Prozess läuft wie folgt ab:

1. Die Antwort wird in eine nützliche Datenstruktur verpackt. Damit können alle im Netzwerkumfeld benötigten Graphen abgebildet werden.

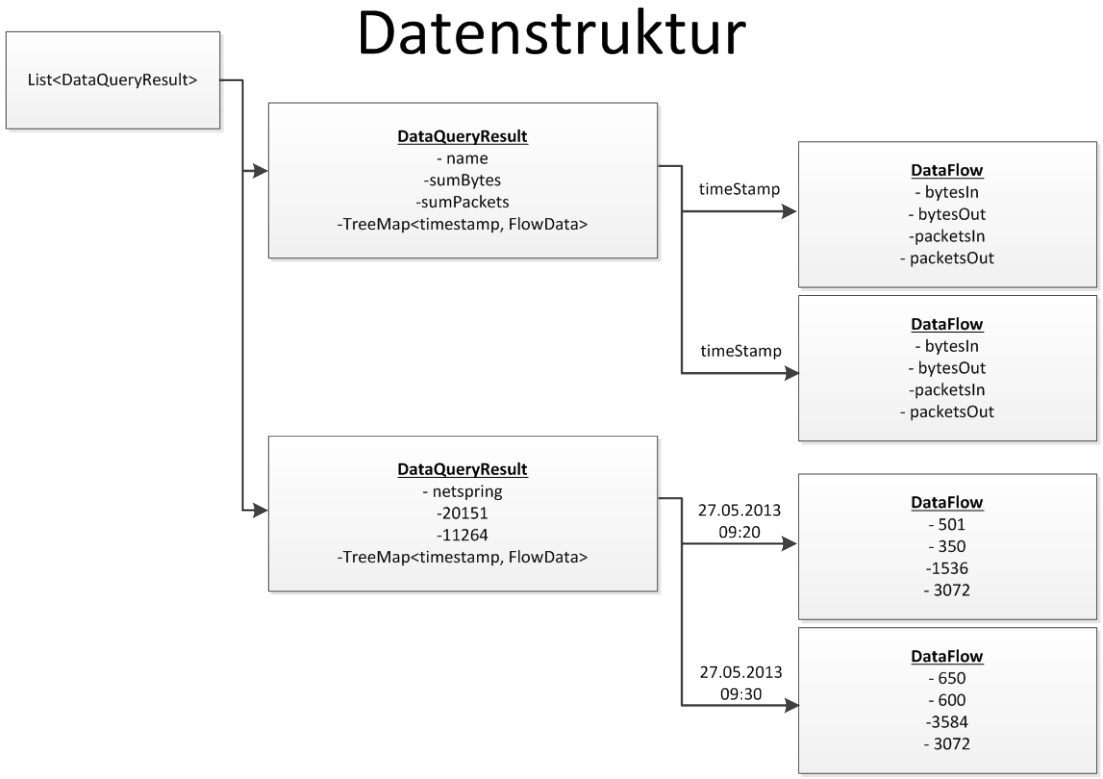


Abbildung 47: Datenstruktur Aufbereitung

2. Werte der Spalten „packet“ und „bytes“ werden jeweils entsprechend umgerechnet. Diese Umrechnung ist im kommenden Abschnitt beschrieben.
3. Die Daten, werden je nach Auswahl nach „packets“ oder „bytes“ absteigend sortiert.
4. Die TopX-Funktion schneidet die sortierte Liste so zu, dass nur die relevanten Daten angezeigt werden. Zusätzlich werden alle abgeschnittenen Werte zusammengefasst und als „All Others“ dargestellt.

5.6.2.1 Einheiten

In der Netzwerkwelt ist es üblich, die Daten in Mbit/s darzustellen. PeeringBox lässt auch andere Einheiten zu. Dazu müssen die summierten Bytes aus der Datenbank jedoch umgerechnet werden. Im folgenden Enum werden die Umrechnungsfaktoren (Divisoren) festgehalten.

```
//            Display-Name   Byte            Bit            Paket
ABSOLUTE(0,   "Absolute",    1,              0.125,         1),
KILO(1,       "Kilo",        1024,           128,           1000),
MEGA(2,       "Mega",        1048576,        131072,        1000000),
GIGA(3,       "Giga",        1073741824,     134217728,     1000000000),
ABSOLUTEPS(4, "Absolute/s",  60,             7.5,           60),
KILOPS(5,     "Kilo/s",      61440,          7680,          60000),
MEGAPS(6,     "Mega/s",      62914560,       7864320,       60000000),
GIGAPS(7,     "Giga/s",      64424509440d,   8053063680d,   60000000000d);
```

Die zeitbezogenen Faktoren sind auf eine Minute genormt. Falls die Daten in einer anderen Granularität vorliegen, wird dies mit beachtet.

5.6.3 Visualisierung

Die Visualisierung der ausgewählten Daten geschieht direkt auf dem Client. Die Daten, welche auf der Webseite dargestellt werden sollen, werden im JSON Format an den Client übermittelt, welcher diese dann anzeigt.

5.6.3.1 Ablauf

Über eine JavaScript Methode, welche beim Laden der Webseite ausgeführt wird, werden die Daten vom Server gelesen. Diese JavaScript Methode konfiguriert zusätzlich den NVD3 Graphen und übergibt diesem die Daten.

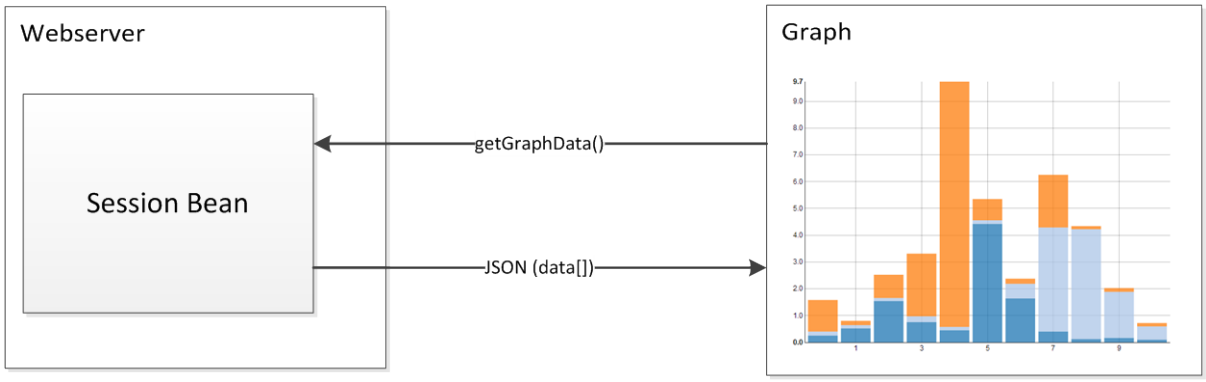


Abbildung 48: Ablaufdiagramm Visualisierung

5.6.3.2 Daten

Von NVD3 werden Daten im JSON Format akzeptiert, welche wir aufbereiten und an den Graphen übergeben werden. Objekte mit demselben Key werden mit derselben Farbe dargestellt. Negative Daten werden also mit demselben Key in der gleichen Reihenfolge unter den Positiven angehängt.

```
[
  {
      "key" : "Object1" ,
      "values" :
       [
                  [ timestamp1 , data1] ,
                  [ timestamp2 , data2] ,
                  [ timestamp3 , data3]
       ]
   }
   {
      "key" : "Object1" ,
      "values" :
       [
                  [ timestamp1 , -data1] ,
                  [ timestamp2 , -data2] ,
                  [ timestamp3 , -data3]
       ]
   }
]
```

5.6.3.3 NVD3

Um die Daten auf der Webseite darzustellen, verwenden wir die JavaScript Library NVD3, welche auf d3.js basiert. NVD3 stellt verschiedene Standardgraphen als Models zur Verfügung. Wir verwenden für unsere Zwecke das Stacked Multi-Bar Model:

- http://nvd3.org/ghpages/multiBar.html

Dieses Model haben wir für unsere Zwecke und Anforderungen angepasst. Dazu gehören unter anderem Änderungen an der Legende und an der Anordnung der Bars.

Durch die Library haben wir den Vorteil, dass der Graph von allen gängigen Browsern unterstützt wird. Die Berechnung des Graphen beim Client bietet den Vorteil, dass mit den Elementen des Graphen interagiert werden kann. Zum Beispiel Tooltip und Ein-/Ausblenden von Objekten beim Klick auf die Legende sind möglich.

Bei grossen und dynamischen Datenmengen verwendet die JavaScript Library clientseitig ziemlich viele Ressourcen. Die Verwendung von Top10 ist zu empfehlen.

5.7 Sicherheit

5.7.1 https

Der Traffic zwischen Webbrowser und Server ist stets verschlüsselt. Ein Mittelsmann, der den Verkehr mitliest ist nicht im Stande die Daten, welche ausgetauscht werden, mitzuhören. Somit werden weder Login Daten noch Auswertungen jemals unverschlüsselt übermittelt.

GlassFish unterstützt die SSL 3.0 und TLS 1.0 als Encryption-Standard. Aktuell sind keine Angriffe bekannt, die ein Aufbrechen der Verschlüsselung innert nützlicher Frist ermöglichen.

Da das Zertifikat aktuell selbst signiert ist, erscheint bei der ersten Anmeldung an den Dienst eine Fehlermeldung. Im Weiteren erscheint im Browser ein Symbol, das zeigt, dass das Zertifikat von keiner „trusted CA“ validiert wurde. Im produktiven Einsatz ist die Verwendung eines „offiziellen“, signierten Zertifikats zu empfehlen.

5.7.2 Interne Passwörter

Schwache Passwörter stellen immer ein Risiko dar. Aus diesem Grund wurde auch intern Wert darauf gelegt, dass Passwörter sicher sind.

Interne Passwörter werden persönlich übergeben.

5.7.3 GlassFish - Admin

Die Adminkonsole des Anwendungsservers wird im Secure-Mode betrieben. Damit wird ein Zugriff nur über https und über die Eingabe eines Passworts erlaubt. Zusätzlich ist die Adminkonsole nicht öffentlich zugänglich (kein offener Port). Lediglich via SSH-Tunnel ist es möglich auf diesen Dienst zuzugreifen. Es muss also zuerst ein Login am Server erfolgen, um überhaupt auf die Glassfish-Konsole zugreifen zu können.

5.7.4 Offene Ports

Die folgenden Ports sind von ausserhalb erreichbar.

Port	Service	Einschränkungen
22	SSH (Shell)	Lediglich über das Schulnetz der HSR erreichbar und nur über Linux-Benutzername / Passwort möglich. Root kann sich nicht direkt einloggen.
80	http (Webservice)	Lediglich um eine Umleitung auf https zu erreichen.
443	https (Webservice)	uneingeschränkt
6330-6343	sFlow-Collector (sFlow-Daten der Switches)	Nur aus IX-Netzwerk erreichbar

Aktuell sind auch Port 80 und Port 443 lediglich über das Schulnetz zu erreichen. Im produktiven Einsatz, wird dies nicht mehr der Fall sein.

5.7.5 Interne Ports

Sofern dies von aussen erfolgen soll, dann muss dies durch einen SSH-Tunnel erfolgen.

Port	Service	Einschränkungen
4848	Glasfish – Adminkonsole	Lediglich über GlassFish-Benutzer „admin“ möglich.
5432	PostgreSQL Datenbanksystem	Lediglich über Benutzer peeringbox und postgres möglich

5.7.6 Userpasswörter

5.7.6.1 In der Datenbank

Als Betreiber einer öffentlichen Anwendung hat man in der heutigen Zeit eine Verpflichtung dem Benutzer gegenüber. Faulheit liegt leider in der Natur des Menschen, weshalb viele Benutzer einerseits einfache Passwörter wählen und andererseits dieselben Passwörter für diverse Dienste verwenden. Die Kombination aus E-Mail-Adresse und Passwort ermöglicht oftmals auf diversen Seiten Zugriff.

Aus diesem Grund werden sämtliche Passwörter zuerst gehasht (zerhacken, verschleiert) und erst dann in der Datenbank gespeichert. Dieses „hashen“ wird mit Hilfe des SHA-256-Alogrithmus sichergestellt. Dieser Algorithmus entspringt der Familie der SHA-2 Algorithmen. Aktuell sind keine Attacken gegen diese Familie bekannt. NIST (National Institute of Standards and Technology) empfiehlt aktuell diesen Algorithmus zu verwenden, auch wenn die SHA-3 Familie bereits in den Kinderschuhen steckt.

Da bei der Anmeldung wieder erst das eingegebene Passwort gehasht wird und erst dann mit der Datenbank abgeglichen wird, werden zu keinem Zeitpunkt Klartextpasswörter persistent gehalten. In einem absolut unglücklichen Falle, indem die Datenbank der Applikation abhandenkommt, könnten kriminelle Energien nur mit enormem Aufwand die Passwörter errechnen.

5.7.6.2 Passwortrichtlinien

Damit ein Passwort akzeptiert wird, muss es folgende Mindestanforderungen erfüllen:

- Mind. 1 Zahl enthalten
- Mind. 1 Kleinbuchstaben enthalten
- Mind. 1 Grossbuchstaben enthalten
- Mind. 6 Zeichen lang sein

Sonderzeichen sind erlaubt, jedoch nicht zwingend erforderlich. Die Richtlinien wurden für genügend stark erachtet. Die folgende Aufstellung unterstreicht diese Aussage.

Die Anzahl aller Kombinationen kann errechnet werden, in dem die Anzahl der Zeichen im Alphabet hoch der Länge des Passworts gerechnet wird.

Zahlen:	10 Zeichen
Kleinbuchstaben:	26 Zeichen
Grossbuchstaben:	26 Zeichen
Summe:	62 Zeichen

Dies entspricht also 62^6 = 56‘800‘235‘584 Kombinationen.

Diese Richtlinie wird mit Hilfe einer regular expression (RegEx) sichergestellt. Eine Anpassung der Richtlinien ist also ohne grossen Aufwand möglich.

5.7.6.3 „Passwort-Recovery“-Funktion

Falls ein Benutzer sein Passwort vergessen haben sollte, kann er dieses selbstständig wiederherstellen. In diesem Abschnitt wird beschrieben, wie dies von statten geht und wie die Implementation realisiert wurde.

5.7.6.3.1 Funktionsweise

Nach einem fehlgeschlagenen Anmeldeversuch wird dem Benutzer die Möglichkeit angezeigt, sein Passwort wiederherzustellen. Die Maske verlangt lediglich die E-Mail-Adresse, an die dann eine E-Mail versandt wird, in der alle notwendigen Informationen enthalten sind.

Natürlich bekommt der Benutzer auch weiterhin die Möglichkeit, sich einzuloggen.

Da sich diese Maske im öffentlichen Bereich der Seite befindet, wurde dazu ein eigenes Bean entwickelt, das lediglich die Funktionalität der Passwortwiederherstellung beheimatet. Dies garantiert, dass von aussen keinerlei Rückschlüsse auf den Rest des Systems auszumachen sind.

Folgende Bedingungen gelten, damit eine E-Mail verschickt wird.

- Die Eingabe muss nach gültigem Format eingegeben werden.
- Die E-Mail-Adresse muss im System erfasst sein.

Das Passwort wird von einem Zufallsgenerator erstellt. Dabei wird bewusst ein kryptisches Passwort mit vielen Zeichen generiert. Das soll den Benutzer dazu bewegen, das Passwort nach Erhalt der Mail manuell zu ändern.

5.7.6.3.2 Eingabemaske

Die Eingabemaske schaut wie folgt aus:

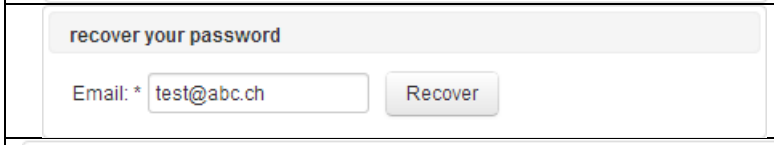
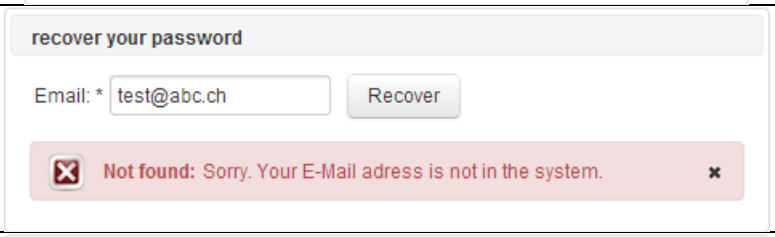
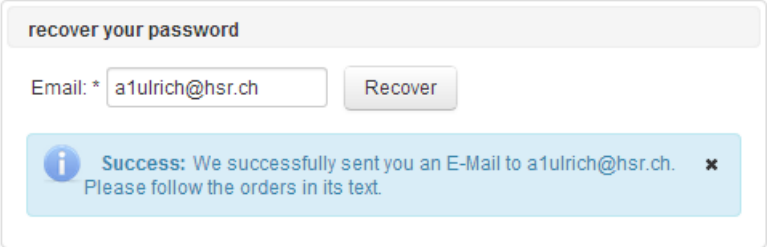
	Solange die Eingabe nicht dem gültigen Format entspricht, wird ein rotes X angezeigt.
	Sobald die Eingabe korrekt ist, wird kein Fehler angezeigt.
	Falls sich die E-Mail-Adresse nicht im System befindet, wird diese Fehlermeldung nach dem Klick auf „Recover“ angezeigt.
	Im Falle eines erfolgreichen Versands, wird diese Meldung angezeigt.

Abbildung 49: Passwortwiederherstellung

5.7.6.3.3 E-Mail Body

In der versandten Mail steht folgender Text:

> Hello username,
>
> This is your new password: eh9ejbg0je46ji64j15j0198agc0b0
>
> You can log into your account with your username "username" and the newly created password above.
>
> Please set a new password as soon as possible. To do so, go to your profile settings.
>
> If there are any problems feel free to contact us.

Legende:
- Passwort Passwort, das per Zufallsgenerator erstellt wird.
- Username: Jeweiliger Username zur zugehörigen E-Mail-Adresse.

Beispiel:

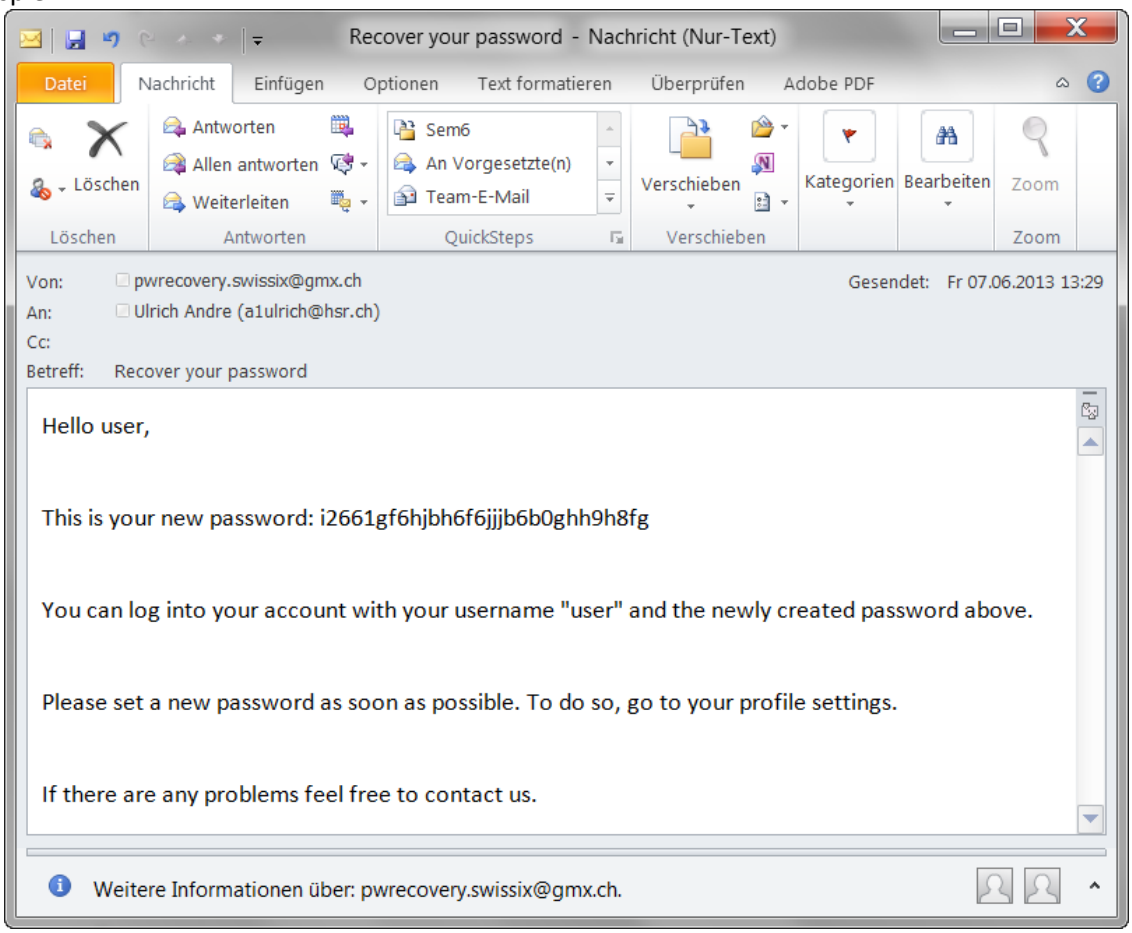


Abbildung 50: Passwortwiederherstellung E-Mail

5.7.6.3.4 E-Mail Header

Im GlassFish kann die Absender-Adresse konfiguriert werden. Wenn man E-Mails direkt vom eigenen Server aus versendet, entsteht ein erhöhter Pflegeaufwand. Man muss sich darum kümmern, dass der Server auf keinen Backlists erscheint und somit nicht als Spam gekennzeichnet, oder gar gelöscht wird. Aus diesem Grund haben wir uns dazu entschiedenen, die Mails über einen externen Provider zu verschicken. Dieser Entscheid ist aber im Mail-Header erkennbar. Unten ist nun ein Auszug des Mail-Headers zu finden, um die Thematik zu veranschaulichen.

```
Received: from mx1.hsr.ch (152.96.21.55) by smtp.hsr.ch (152.96.21.200) with  Microsoft SMTP Server id
        14.3.123.3; Fri, 31 May 2013 08:24:32 +0200
Received: from localhost (localhost [127.0.0.1])       by mx1.hsr.ch (Postfix) with ESMTP id 11B7CA6F1
        for <receiver@hsr.ch>; Fri, 31 May 2013 08:24:32 +0200 (CEST)
X-Quarantine-ID: <QNkoBlOPoeKz>
X-Spam-Flag: NO
X-Spam-Score: 0.1
X-Spam-Level:
X-Spam-Status: No, score=0.1 tagged_above=-999 required=4.8 tests=[L_P0F_UNKN=0.1] autolearn=no
X-Amavis-OS-Fingerprint: UNKNOWN [S4:55:1:60:M1380,S,T,N,W9:.:?:?] (NAT!) (up: 1478 hrs), (link: GPRS, T1,
        FreeS/WAN), [212.227.17.22:58047]
Received: from mx1.hsr.ch ([127.0.0.1])      by localhost (mx1.hsr.ch [127.0.0.1]) (amavisd-new, port 10024)
        with LMTP id QNkoBlOPoeKz for < receiver@hsr.ch >; Fri, 31 May 2013 08:24:31 +0200 (CEST)
Received: from mout.gmx.net (mout.gmx.net [212.227.17.22])          by mx1.hsr.ch
        (Postfix) with ESMTP id D3D3AA0E6          for <receiver @hsr.ch>; Fri, 31 May 2013
        08:24:31 +0200 (CEST)
Received: from mailout-de.gmx.net ([10.1.76.24]) by mrigmx.server.lan  (mrigmx001) with ESMTP (Nemesis)
        id 0Ljwk3-1U7KJa2Z4L-00bomk for  <a1ulrich@hsr.ch>; Fri, 31 May 2013 08:24:31 +0200
Received: (qmail invoked by alias); 31 May 2013 06:24:31 -0000
Received: from sflow.swissix.ch (EHLO sflow.swissix.ch) [194.242.34.154] by  mail.gmx.net (mp024) with SMTP;
        31 May 2013 08:24:31 +0200
X-Authenticated: #164278387
X-Provags-ID: V01U2FsdGVkX1+sDD08yqusvMnezVrrpbcvPVm7M+I2EdSCBMMBftl4n9fR2GWz3JTQ
To: Reveiver Firstname Lastname <receiver@hsr.ch>
Message-ID: <1691067548.0.1369981470597.JavaMail.pwrecovery.swissix@gmx.ch>
Subject: Recover your E-Mail
MIME-Version: 1.0
Content-Type: text/plain; charset="us-ascii"
Content-Transfer-Encoding: 7bit
X-Y-GMX-Trusted: 0
Date: Fri, 31 May 2013 08:24:32 +0200
From: <pwrecovery.swissix@gmx.ch>
Return-Path: pwrecovery.swissix@gmx.ch
X-MS-Exchange-Organization-AuthSource: SID00200.hsr.ch
X-MS-Exchange-Organization-AuthAs: Anonymous
```

Legende:

- GMX-Hinweise	Absenderadresse:	pwrecovery.swissix@gmx.ch
- Empfängerbezogen	Empfängeradresse:	receiver@hsr.ch
- SwissIX	Servername:	sflow.swissix.ch

In Absprache mit SwissIX ist die Verwendung von GMX im Rahmen der Bachelorarbeit erlaubt. Zu gegebener Zeit wird dies aber angepasst.

5.7.7 ReverseProxy

Im Rahmen einer Abklärung wurde untersucht, wie die Sicherheit der Anwendung erhöht würde, wenn ein Reverseproxy vor die Anwendung geschaltet würde.

Verletzlichkeiten von Glassfish, bzw. dessen Teilkomponenten (Apache Tomcat mit Grizzly) sind von aussen unerreichbar. Um diese Schwachstellen ausnützen zu können, müsste erst eine Verletzlichkeit im Reverseproxy gefunden werden. Dieses „Abstraktions-Layer“ erschwert also einen Angriff auf den Server.

Keinen Schutz bietet dieses Abstraktions-Layer jedoch vor Angriffsmethoden, wie SQL-Injection, denn diese Statements werden direkt an die Applikation weiter geleitet.

Des Weiteren werden laut Recherchen mehr 0Day-Exploits für breit eingesetzte Applikationen wie Apache oder NginX (die gut als Reverseproxy eingesetzt werden könnten) gefunden, als dies bei GlassFish der Fall ist. Sobald der Angreifer root-Zugriff auf den Server erlangt hat, nützt auch das zusätzliche Abstraktionslayer nichts mehr. Sobald diese zwei Komponenten getrennt auf zwei Servern betrieben werden, wäre definitiv ein erhöhter Schutz vorhanden. Dies muss jedoch vom jeweiligen Betreiber der Applikation individuell entschieden werden. Als Folge entständen eine erhöhte Last und ein grösserer administrativer Aufwand.

Quellen:

- http://blog.eisele.net/2011/05/securing-your-glassfish-hardening-guide.html
- http://trends.builtwith.com/framework/J2EE
- http://trends.builtwith.com/Web%20Server/top
- Apache » Http Server : Security Vulnerabilities (172)
  - http://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/Apache-Http-Server.html
- Oracle » GlassFish Server : Security Vulnerabilities (14)
  - http://www.cvedetails.com/vulnerability-list/vendor_id-93/product_id-20700/Oracle-Glassfish-Server.html

5.7.8 Datensicherung / Wiederherstellung

Die Applikation kann mit den Dateien, die zusammen mit diesem Dokument abgegeben wurden, stets wiederhergestellt werden. Natürlich sind im Falle eines Systemausfalles aber alle IX-bezogenen Informationen verloren. Diese gilt es auch zu sichern. Dieser Abschnitt befasst sich genau mit dieser Thematik.

Die folgenden Daten gilt es zu sichern, damit auch gesammelte Daten wiederhergestellt werden können.

- Benutzer
- Benutzerrollen
- Peers
- Benutzerzugehörigkeiten (Rollen und Peers)
- Ports
- PortAllocations
- sFlow-Daten

Sämtliche Daten finden sich auf dem Datenbankserver wieder und bilden den gesamten Inhalt der Datenbank ab.

5.7.8.1 Systemzustandsinformationen

Bei Systemzustandsinformationen handelt es sich um alle Daten, die den Zustand des Systems zu einem gewissen Zeitpunkt abbilden. Dies sind die oben aufgeführten Daten, mit Ausnahme der sFlow-Daten. Da es sich dabei nur um sehr kleine Datenmengen handelt, bietet sich eine komplette Sicherung der einzelnen Tabellen an. Da sich diese auch nur selten ändern, ist hier ein täglicher Dump des gesamten Inhalts zu empfehlen.

Unter PostgresSQL kann ein Dump einer einzelnen Tabelle wie folgt ausgeführt werden.

```
$ pg_dump -t mytab mydb > tableBackup.sql
```

Dies kann per Script automatisiert werden. Konkrete Informationen zu Datenbank-Dumps im Zusammenhang mit PostgreSQL sind hier zu finden:

- http://www.postgresql.org/docs/9.1/static/backup.html
- http://www.postgresql.org/docs/9.1/static/backup-dump.html
- http://www.postgresql.org/docs/9.1/static/app-pgdump.html

5.7.8.2 sFlow-Daten

Die sFlow-Daten können unmöglich täglich komplett gesichert werden, da dabei mehrere 100 GB gesichert werden müssten. PostgreSQL bietet hier bequeme Möglichkeiten um mit Scripts automatisiert nur die gewollten Daten zu sichern. Unsere Vorschläge lauten wie folgt:

Möglichkeit 1 – Dump

Ein Script, das die neu aggregierten Daten als Dump sichert, wäre möglich. Dieses könnte je nach Wichtigkeit die Tages-, oder gar Monatsdaten partiell sichern. Da alte Daten zu keinem Zeitpunkt modifiziert werden, wäre damit auch die Konsistenz der Daten garantiert.

Vorteil:

- Geregelter Ablauf
- Ressourcenschonend

Nachteile:

- Im schlimmsten Fall geht ein kompletter Tag/Monat verloren.

Möglichkeit 2 - Replikation

PostgreSQL-Datenbanken können zwischen verschiedenen Servern repliziert werden. Damit würde man auch die Notwendigkeit eines Backups verringern, da sämtliche Daten bereits an mehreren Orten vorhanden wären.

Vorteile

- Im Falle eines Ausfalls keinerlei Datenverlust
- Mögliche Lastaufteilung bei Abfragen
- Kein Aufwand für Backup

Nachteile:

- Kosten für zusätzlichen Server
- Viel Traffic zwischen den Servern
- Grössere Datenmenge

Die Entscheidung ist dem IX überlassen. Eine Kombination der Möglichkeiten wäre ebenfalls denkbar. Bei einer Datensicherung muss immer darauf geachtet werden, dass sie nicht auf dieselbe Hardware gemacht wird.

5.8 Tests

5.8.1 Unit Tests

Der Screenshot zeigt die JUnit-Tests, die während der Entwicklung erstellt wurden. Diese Tests decken alle öffentlichen (public) Methoden ab. Ausgenommen davon sind Klassen mit Datenbankanbindung, oder userinterfacespeifische Klassen.

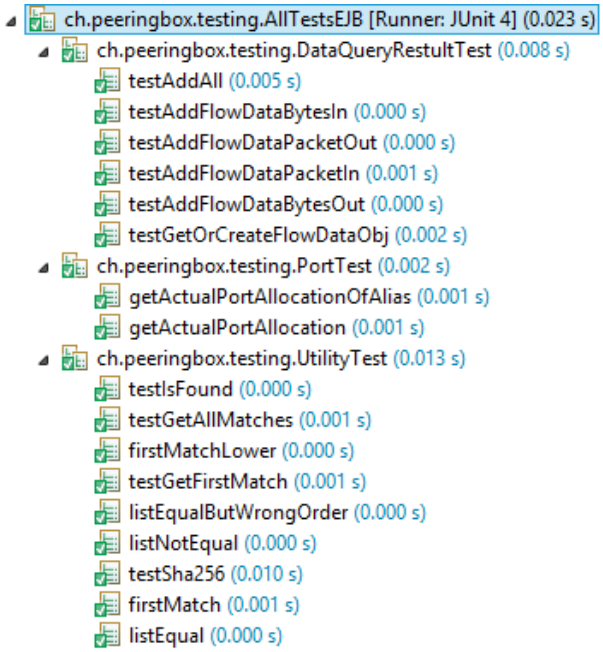


Abbildung 51: JUnit-Tests EJB

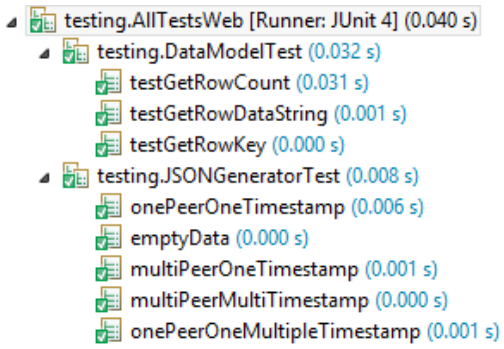


Abbildung 52: JUnit-Tests Web

5.8.2 Funktionale Tests

Untenstehende Tabelle zeigt die Checkliste, welche nach allen Änderungen am Livesystem geprüft wurde, um sicherzustellen, dass keine offensichtlichen Fehler veröffentlicht werden.

Bezeichnung	Beschreibung	Status
sFlow Sammlung	Tabellen der verschiedenen Stunden werden in der Datenbank erstellt und enthalten die erwartete Anzahl an Einträgen.	✓
sFlow Historisierung	Tabellen der verschiedenen Tage Monate und Jahre werden in der Datenbank erstellt und enthalten die erwartete Anzahl an Einträgen. Gleichzeitig werden aggregierte Tabellen gelöscht und sind dann nicht mehr im System vorhanden.	✓
SNMP-Crawler (Daten)	Erstellte Ports, PortAllocations, Peers und Macadressen sind gemäss den SNMP-Daten erstellt worden.	✓
SNMP-Crawler (Änderungen)	Änderungen werden vom System innert des definierten Intervalls erkannt und im System festgehalten.	✓
Login	Das Login funktioniert und der richte Benutzer wird authentisiert. Falsche Passwörter resultieren in einem Fehler.	✓
Benutzer (CRUD)	Benutzer können im System erfasst, gelöscht und mutiert werden.	✓
Selektionen	Die getätigten Selektionen werden richtig übernommen.	✓
Graphen	Graphen werden erstellt und korrekt visualisiert	✓
Integrität	Es werden lediglich Daten angezeigt, die einem Peer angehören, der auch dem Benutzer zugeordnet ist.	✓
Passwort zurücksetzen	Im Falle eines vergessen geratenen Passworts kann über die „Recover“-Funktion ein neues Passwort generiert werden. Dieses wird per Mail an den entsprechenden Benutzer versandt und wird darauf hin beim Login akzeptiert.	✓
Alarm(CRUD)	Alarme können erstellt und gelöscht werden. (nicht implementiert)	✗
Alarm (Auslösung)	Alarme werden periodisch überprüft. Ungereimtheiten werden per Mail dem entsprechenden Benutzer gemeldet. (nicht implementiert)	✗

5.9 Projektstruktur

Das Projekt ist in zwei Teile gegliedert. Der Business Layer wird durch das Projekt peeringboxEJB abgebildet.

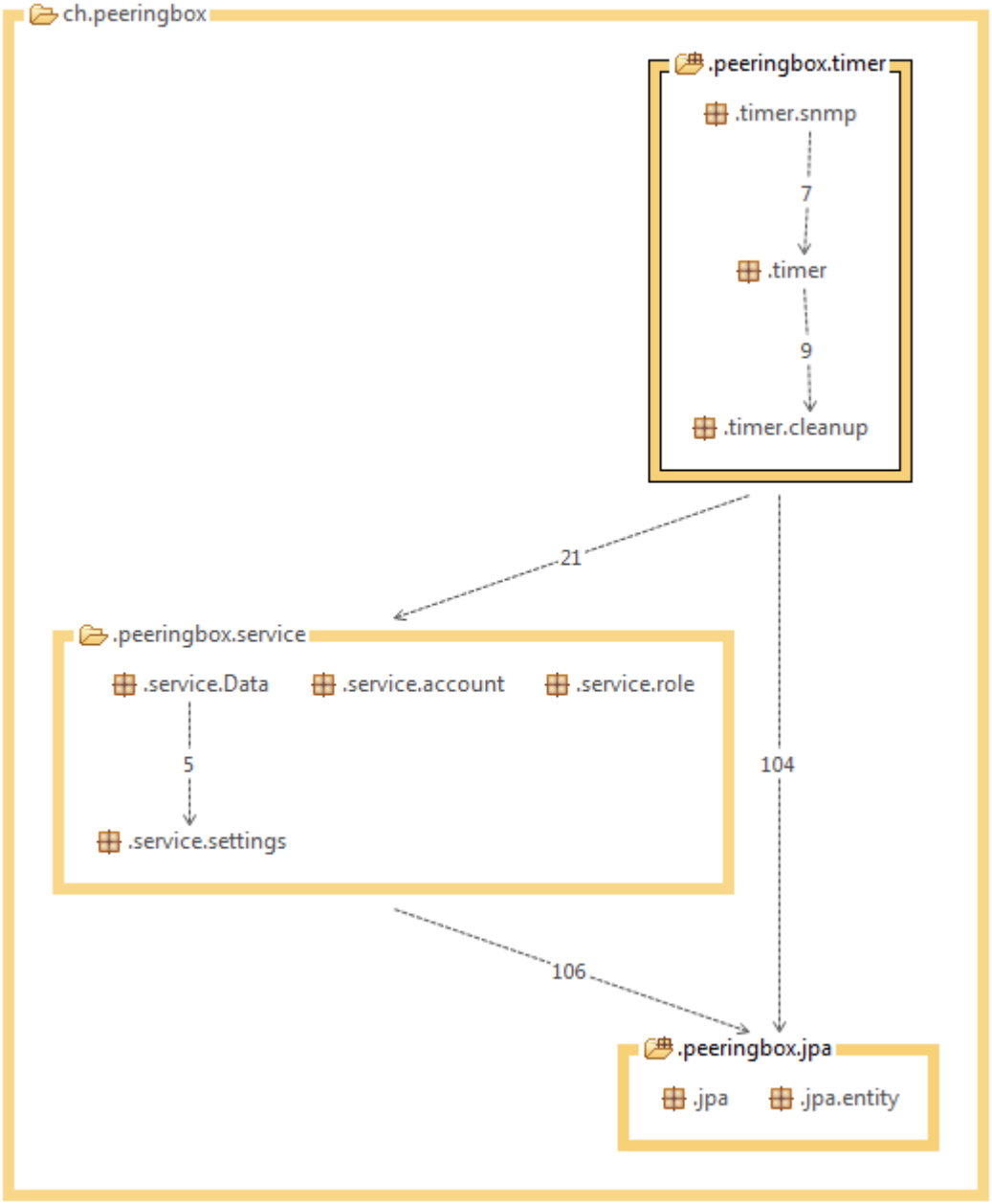


Abbildung 53: Projektstruktur peeringboxEJB

Der Webserver wird durch das Projekt peeringboxWeb abgebildet.

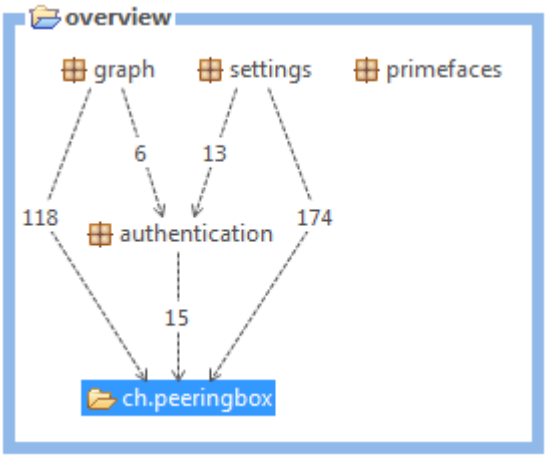


Abbildung 54: Projektstruktur peeringboxWeb

5.10 Metrik

Die PeeringBox in Zahlen ausgedrückt. Dabei wird nur Java-Code beachtet. JavaScript und xHtml zählt nicht zur Statistik.

5.10.1 Business Layer (peeringboxEJB)

Anzahl Packages	11
Anzahl Classes	51
Anzahl Methods	453
Total Codezeilen	3377

5.10.2 Webseite (peeringboxWeb)

Anzahl Packages	6
Anzahl Classes	27
Anzahl Methods	216
Total Codezeilen	1590

5.10.3 Summe

Anzahl Packages	17
Anzahl Classes	78
Anzahl Methods	669
Total Codezeilen	4960

6. Glossar

Der Einfachheit halber werden im Dokument Abkürzungen verwendet. Diese werden hier kurz erläutert.

Abkürzung	Bedeutung
IX	Internet Exchange
SA	Studienarbeit
BA	Bachelorarbeit
JSF	JavaServer Faces
JS	JavaScript
JPA	Java Persistence API
JTA	Java Transaction API
ISP	Internet Service Provider: stellt Internet Service zur Verfügung (z.B.: Swisscom, Cablecom, etc.)
IXP	Internet Exchange Point: Ein Knoten der es erlaubt Internet Verkehr zwischen ISP's auszutauschen.
Peer	Ein Endpunkt einer Kommunikation in einem Computernetzwerk.

7. Abbildungsverzeichnis