HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

COIViz

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2011

Autor:	Marion Frei
Betreuer:	Prof. Eduard Glatz
Projektpartner:	-
Experte:	-
Gegenleser:	-

Erklärung über die eigenständige Arbeit

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.

Ort, Datum: Rapperswil, 13.12.2011

Name, Unterschrift:

Marion Frei

Abstract

Innerhalb eines Netzwerks haben Benutzer oft einige gemeinsame Interessen (COI, Community of Interests) in der Form, dass sie dieselben Server und Dienste nutzen. Mit der Entwicklung von COIViz sollen diese COIs visualisiert werden. Dabei wurde bereits nach kurzer Zeit eine experimentelle Richtung eingeschlagen. Es sollten die Möglichkeiten und Grenzen der Visualisierung dieser COIs erforscht werden.

Aus den Flow-Daten, in denen nur die Headerinformationen (Source-IP, Destination-IP, Protokoll, …) gespeichert sind, wird ein bipartiter Graph gebildet, der die Verbindungen zwischen internen und externen IP-Adressen speichert. Dabei wird lediglich der bidirektionale Netzwerkfluss berücksichtigt. Aus dem bipartiten Graph wird ein sogenannter Nachbarschaftsgraph kreiert. Die Knoten im Nachbarschaftsgraph stellen dabei die internen IP-Adressen dar. Zwischen zwei Knoten gibt es eine Kante, falls die zwei Knoten ein oder mehrere gleiche externe IP-Adressen besucht haben. Als Kanten-Notation wird dabei die Anzahl gemeinsam besuchter externer IP-Adressen verwendet.

Dieser Nachbarschaftsgraph kann mit Hilfe des Tools GraphViz[1] und einem kräftebasierten Layout so visualisiert werden, dass Gruppen erkennbar werden. Bei dem kräftebasierten Layout stossen sich die Knoten umso stärker ab, je schwächer ihre Verbindung ist. Dabei hat sich gezeigt, dass neben echten Interessensgruppen "Pseudo-COIs" entstehen. Diese haben untereinander keine gemeinsamen Interessen, hinterlassen jedoch diesen Eindruck. Diese "Pseudo-COIs" entstehen, da die einzelnen Knoten jeweils einige wenige Kanten in Richtung Zentrum des Graphen, jedoch nicht untereinander haben. Durch das kräftebasierte Layout stossen sich diese Knoten stark ab, wobei ihre Distanz zur Mitte des Graphen durch die Kanten vorgegeben ist. Dies führt zu einer ringförmigen Gleichverteilung.

Aus den Visualisierungen und den auftretenden "Pseudo-COIs" folgt, dass dieser Ansatz der Visualisierung nicht zum gewünschten Ziel führt und eine Implementation eines grafischen User Interface deshalb nicht sinnvoll ist.

Management Summary

Ausgangslage

Innerhalb eines Netzwerks haben Benutzer oft einige gemeinsame Interessen (COI, Community of Interests) in der Form, dass sie dieselben Server und Dienste nutzen. Mit der Entwicklung von COIViz sollen diese COIs visualisiert werden. Dabei wurde bereits nach kurzer Zeit eine experimentelle Richtung eingeschlagen. Es sollten die Möglichkeiten und Grenzen der Visualisierung dieser COIs erforscht werden.

Vorgehen

Aus den Netzwerkdaten (sog. Flow-Daten), in denen nur noch die Headerinformationen (Source-IP, Destination-IP, Protokoll, …) gespeichert sind, wird ein bipartiter Graph gebildet, der die Verbindungen zwischen internen und externen IP-Adressen speichert. Dabei wird lediglich der bidirektionale Netzwerkfluss berücksichtigt. Aus dem bipartiten Graph wird ein sogenannter Nachbarschaftsgraph kreiert. Die Knoten im Nachbarschaftsgraph stellen dabei die internen IP-Adressen dar. Zwischen zwei Knoten gibt es eine Kante, falls die zwei Knoten ein oder mehrere gleiche externe IP-Adressen besucht haben, diese externen IP-Adressen also den Interessen von beiden entsprechen. Als Kanten-Notation wird dabei die Anzahl gemeinsam besuchter externer IP-Adressen verwendet.

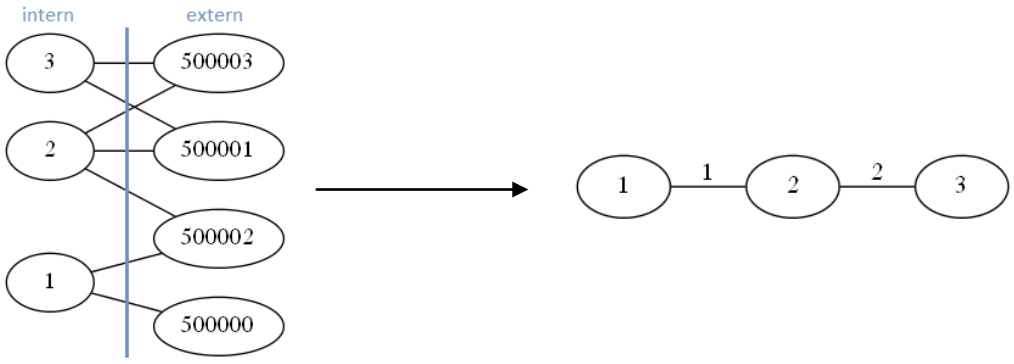


Abbildung 1: Umwandlung von Bipartitem Graph zu Nachbarschaftsgraph

Dieser Nachbarschaftsgraph kann mit Hilfe des Tools GraphViz[1] und einem kräftebasierten Layout so visualisiert werden, dass Gruppen erkennbar werden. Bei dem kräftebasierten Layout stossen sich die Knoten umso stärker ab, je schwächer ihre Verbindung ist.

Ergebnisse

Während der Arbeit hat sich gezeigt, dass neben echten Interessensgruppen "Pseudo-COIs" entstehen. Diese haben untereinander keine gemeinsamen Interessen, hinterlassen jedoch diesen Eindruck. Diese "Pseudo-COIs" entstehen, da die einzelnen

Knoten jeweils einige wenige Kanten in Richtung Zentrum des Graphen, jedoch nicht untereinander haben. Durch das kräftebasierte Layout stossen sich diese Knoten stark ab, wobei ihre Distanz zur Mitte des Graphen durch die Kanten vorgegeben ist. Dies führt zu einer ringförmigen Gleichverteilung.

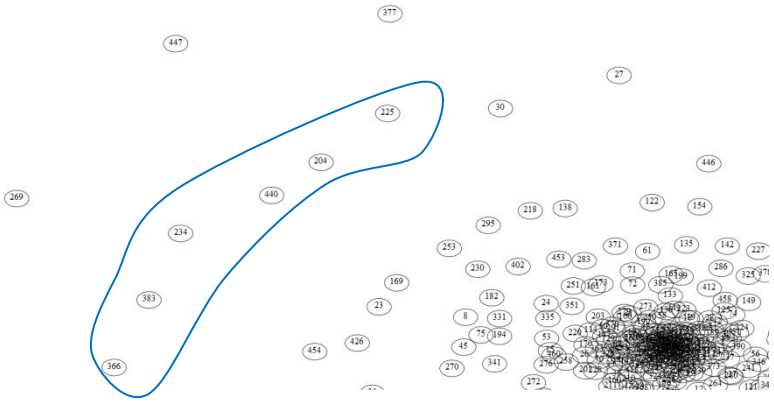


Abbildung 2: Pseudo-COI ohne Kanten

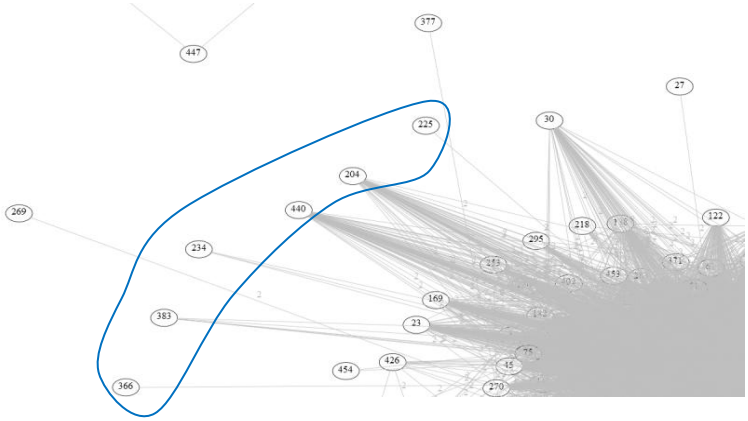


Abbildung 3: Pseudo-COI mit Kanten

Aus den Visualisierungen und den auftretenden "Pseudo-COIs" folgt, dass dieser Ansatz der Visualisierung nicht zum gewünschten Ziel führt und eine Implementation eines grafischen User Interface deshalb nicht sinnvoll ist.

Inhaltsverzeichnis

Teil I

Technischer Bericht

1. Einleitung

Wir alle begeben uns jeden Tag zig Mal ins Web. Dabei surfen wir auf Websites, schreiben Mails und nutzen andere Webservices. Endbenutzer im selben Subnetz weisen dabei teilweise überschneidende Interessen auf. Aus diesen Interessen lassen sich Interessensgruppen (COI, Communities of Interests) identifizieren. Diese Studienarbeit soll diese COIs visualisieren und ihre Möglichkeiten und Grenzen herausfinden.

Als Grundlage um die COIs zu analysieren, stand an der HSR eine spezielle Infrastruktur zur Verfügung ("Analyzer"), die es erlaubte, sämtliche Internet-Verkehrsverbindungen der HSR zu analysieren. Die IP-Adressen der Daten, die der Analyzer aufzeichnet, werden anonymisiert. Um zu sehen, was hinter den gemeinsamen Interessen steht, sollten DNS-Lookups gemacht werden. Dies war mit den anonymisierten Daten nicht möglich.

Dies konnte umgangen werden, indem Zugriff auf den Scylla-Cluster der ETH Zürich gewährt wurde. Über diesen kann der gesamte Verkehr der Schweizer Hochschulen analysiert werden, wobei auch hier nur Daten der HSR analysiert wurden.

Mitschnitte dieser Verkehrsdaten vom Analyzer und von Scylla-Cluster, sogenannte CFlow Dateien, sowie die zur Verarbeitung dieser Dateien nötigen Klassen wurden von Projektbetreuer, Prof. Eduard Glatz zur Verfügung gestellt. Lediglich die Filterung der Dateien vom Scylla-Cluster nach Daten von der HSR musste noch erstellt und herausgefiltert werden.

2. Analyse

2.1. Allgemeine Beschreibung

2.1.1. Produkt Perspektive

Siehe Kapitel 7, Aufgabenstellung.

2.1.2. Benutzer Charakteristik

COIViz wird Benutzern dienen, welche über Zugriff auf Internet-Verkehrsdaten verfügen und aus diesen die Interessengruppen visualisieren und analysieren möchten. Die Zielgruppe besteht vorwiegend aus erfahrenen Benutzern mit Informatik-Hintergrund.

2.1.3. Einschränkungen

- COIViz wird in C++ unter Linux geschrieben und unter Ubuntu Gnome getestet.
- Die einzulesenden Dateien müssen im Cflow-Format vorliegen.

2.1.4. Annahmen

Keine Annahmen vorhanden.

2.1.5. Abhängigkeiten

Auf dem System muss folgende Software / Libraries vorhanden sein:

- Eclipse IDE for C/C++ Developers
- CUTE C++ Test Framework 4.3.0
- Gcc 4.4.5
- Glib 2.0
- Graphviz 2.26.3-4
- Boost 1.42 und Boost I/OStreams 1.42

2.2. Spezifische Anforderungen

Das experimentelle Ermitteln von geeigneten Visualisierungen des Nachbarschaftsgraphen mit kräftebasiertem Layout und dessen Grenzen stellt das Hauptaugenmerk der Arbeit dar.

2.2.1. Funktionale Anforderungen

- Es können Cflow Verkehrsdaten eingelesen werden.
- Diese werden nach Intern – Extern gefiltert.
- Jeder IP-Adresse wird eine Nummer zugewiesen.
- Es wird ein bipartiter Graph im UV-Format (u,v) erstellt wobei U und V Nummern der Knoten sind und U < V sein muss.

- Der bipartite Graph kann auch nur aus Flow-Daten mit vorgegebenen Ports / Protokollen erstellt werden (Filterung).
- Der bipartite Graph wird in einen Nachbarschaftsgraph umgewandelt. Dort werden Paare von internen Nummern gespeichert, welche eine gemeinsame externe IP-Adresse besucht haben. Haben zwei Nummern mehrere gemeinsame externe IP-Adressen besucht, tritt dieses Paar mehrfach auf.
- Aus dem Nachbarschaftsgraph werden Interessen gebildet. Dort werden die einzelnen Paare zusammen mit der Anzahl ihres Auftretens gespeichert. Jedes Paar in den Interessen ist einmalig.
- Aus den Interessen können Kanten mit wenig Gewicht gelöscht werden.
- Die Interessen werden mit einem kräftebasierten Layout mit und ohne Kanten im Svg-Format visualisiert und auch als Dot-Datei abgespeichert.
- Die externen Knoten werden mit der Anzahl ihres Auftretens gespeichert. Auf den 50 häufigst auftretenden externen Knoten wird ein DNS-Lookup gemacht.

2.2.2. Nichtfunktionale Anforderungen

2.2.2.1. Richtigkeit

Wird eine Datei mit Internet-Verkehrsdaten eingelesen, ist der daraus resultierende Graph stets der gleiche.

2.2.2.2. Benutzbarkeit

Es wird kein GUI realisiert. Das Tool lässt sich von der Kommandozeile starten.

2.2.3. Schnittstellen

2.2.3.1. Benutzerschnittstelle

COIViz lässt sich von der Kommandozeile benutzen.

2.2.3.2. Hardwareschnittstelle

Es wird keine Hardwareschnittstelle benötigt, da COIViz ein reines Software-Projekt ist.

2.2.3.3. Datenbankschnittstelle

Es wird keine Datenbankschnittstelle benötigt, da keine Daten über eine Sitzung hinweg intern gespeichert werden müssen.

3. Design

3.1. Vorgehen

3.1.1. Bipartiter Graph

COIViz liest ein oder mehrere CFlow-Dateien ein. Die darin enthaltenen IP-Adressen müssen vertraulich behandelt werden und dürfen den "Analyzer" nicht verlassen. Um dieser Vorgabe zu entsprechen und die Bearbeitung der Daten zu vereinfachen, wird jeder auftretenden IP-Adresse (Knoten) eine Zahl (Knotennummer) zugewiesen. Die IP-Adresse wird zusammen mit der Nummer in einer Hashmap mit Key (= IP-Adresse) und Value (= Knotennummer) abgelegt. Für das DNS-Lookup muss mit den Knotennummern wieder die IP-Adresse des Knoten aus der Hashmap geholt werden. Da das Suchen mit dem Value nach Key sehr langsam und umständlich ist, wurde extra für das DNS-Lookup eine zweite Hashmap mit vertauschten Key/Value-Paaren angelegt. Dabei erhalten zum Subnetz der HSR gehörende IP-Adressen eine von 1 aufsteigende Nummer während externen IP-Adressen eine hohe Nummer beginnend bei 500'000 zugewiesen wird.

Mit diesen Nummern wird danach der bipartite Graph gebildet. Dieser wird durch ein Hashset repräsentiert und beinhaltet Paare in UV-Form (KnotenNr1, KnotenNr2) welche eine Kante darstellen, wobei KnotenNr1 kleiner als KnotenNr2 ist. Dabei ist das Vorhandensein einer Kante wichtig und nicht die Anzahl ihres Auftretens. Die einzelnen Einträge in dieser Hashset sind deshalb einzigartig.

Paare in UV-Form:	
1	500000
1	500002
2	500001
2	500002
2	500003
3	500001
3	500003

Tabelle1: Einfaches Beispiel einer UV-Form

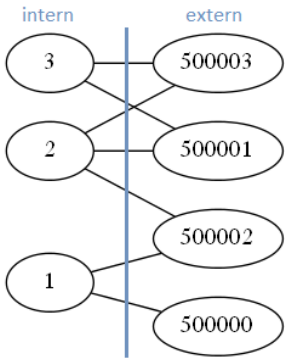


Abbildung 4: Beispiel eines bipartiten Graphen

Der am meisten aufgesuchte, gemeinsame, externe Knoten ist, wie man erwarten würde, der DNS-Server. Dieser ist jedoch nicht von Interesse. Um die DNS-Anfragen herauszufiltern und die Interessensgruppen differenzierter betrachten zu können, wurde ein Filter eingefügt. Dieser filtert die Flows direkt nach dem Einlesen, noch bevor sie verarbeitet werden. Dies erhöht auch die Performance. Die Flows können nach folgenden Ports oder Protokollen gefiltert werden.

- DNS_PORTS: filtert alle Flows mit Port 53 und Port 0. Neben dem DNS-Port werden hier auch fragmentierte Flows gefiltert.

- WEB_PORTS: filtert die Ports 21, 80, 443 und 8080.
- MAIL_PORTS: filtert die Ports 25, 110, 143, 993 und 995.
- FTP_PORT: filtert Port 21.
- HTTP_PORTS: filtert nach Port 80 und 8080.
- ICMP-Protokoll
- TCP-Protokoll
- UDP-Protokoll

3.1.2. Nachbarschaftsgraph

Aus dem bipartiten Graph wird ein Nachbarschaftsgraph generiert. Dabei wird eine leicht abgeänderte Version eines von Prof. Eduard Glatz zur Verfügung gestellten Algorithmus verwendet. Dieser benötigt den bipartiten Graphen in sortierter Form, was ein effizientes Berechnen des Nachbarschaftsgraphen erlaubt.

Der Algorithmus sucht nach internen Knotennummern, welche dieselben externen Knotennummern aufsuchen, und speichert diese paarweise als Kanten. Dabei werden Paare, die mehrere externe KnotenNummern gemeinsamen haben, auch mehrfach gespeichert.

Da während des Projektes die Anforderung dazukam, DNS-Lookups der externen Vertices machen zu können, musste der Nachbarschaftsgraph so umgestellt werden, dass er neben den internen Knotennummern-Paaren auch die gemeinsame externe Knotennummern speichert.

Nachbarschaftsgraph:		
intern	intern	gemeinsamer externer
1	2	500002
2	3	500001
2	3	500003

Tabelle2: einfaches Beispiel eines Nachbarschaftsgraph

3.1.3. Interessen

Danach werden die gemeinsamen Interessen der internen Knotennummern gezählt. Da nun die identischen Kanten im Nachbarschaftsgraph gezählt werden, werden diese nur noch ein Mal gespeichert. Die Anzahl ihres Auftretens wird als Kantengewicht eingeführt. Dabei müssen, wieder wegen des DNS-Lookups, alle gemeinsam aufgesuchten externen Knotennummern mitgespeichert werden.

Interessen:			
intern	intern	Kantengewicht	extern
1	2	1	500002
2	3	2	500001, 500003

Tabelle 3: einfaches Beispiel Interessen

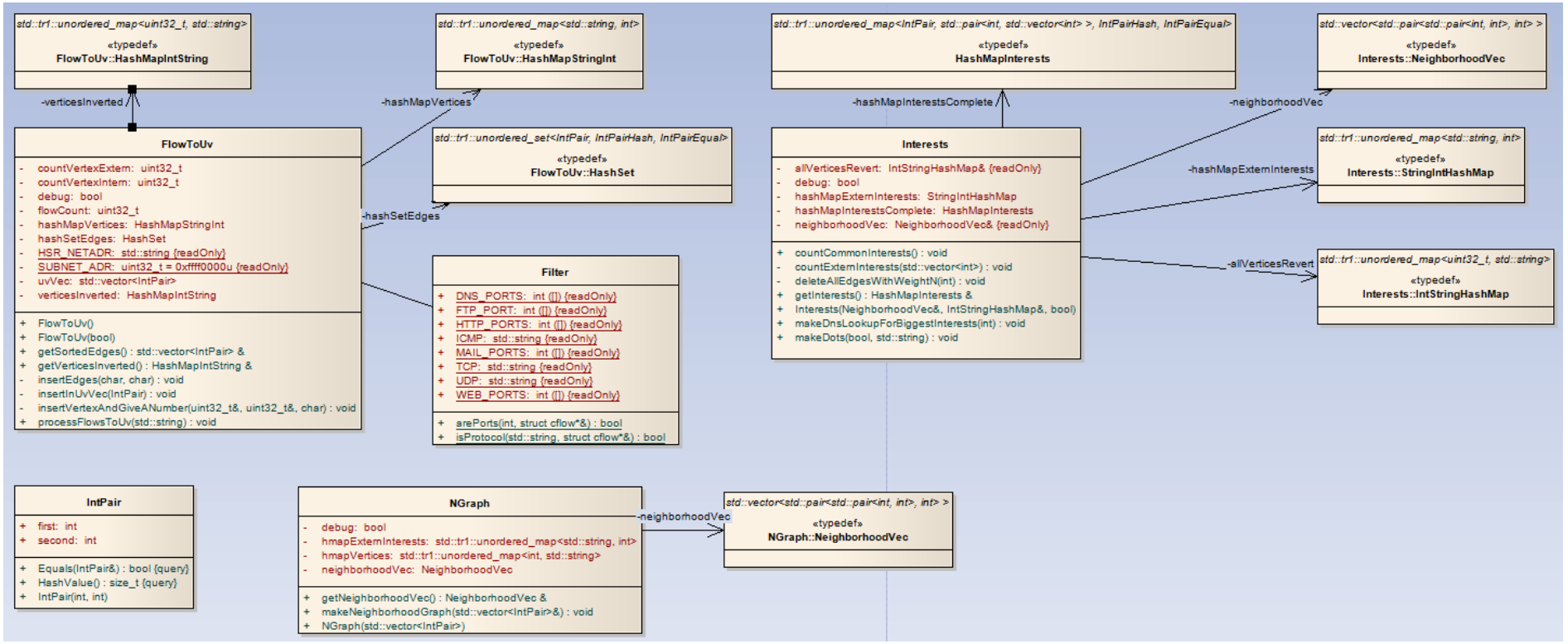
Abbildung 5: Interessen mit Kantengewicht

Da grössere gemeinsame Interessen dargestellt werden sollen, sind Kanten, die nur wenige Male vorkommen, nicht von Interesse. Daher werden alle Kanten, die ein Kantengewicht kleiner oder gleich der Zahl N sind, gelöscht. Dabei kann N desto grösser gewählt werden, umso mehr Daten verarbeitet werden.

Für das DNS-Lookup wird gezählt, welche externen Knoten wie oft vorkommen, d.h gemeinsame Interessen darstellen. Um ein DNS-Lookup machen zu können, muss aus den Knotennummern wieder die IP-Adresse ausgelesen werden. Auf den 50 am meisten vorkommenden externen Knoten wird dann das DNS-Lookup ausgeführt und die Ergebnisse in eine Textdatei geschrieben. Sollte das Lookup zu einer IP-Adresse keinen Namen zurückliefern, wird die IP-Adresse eingesetzt.

Aus den Interessen wird eine Dot-Datei geschrieben, bei dem die Anzahl gemeinsam besuchter externer Vertices als Kantennotation benutzt wird. Diese Dot-Datei wird einmal mit sichtbaren Kanten, einmal ohne Kanten geschrieben. Die Visualisierung der zwei Dot-Dateien wird mit dem kräftebasiertem Neato-Layout des GraphViz Tools gemacht. Das Neato-Layout zeichnet sich dadurch aus, dass sich Vertices je nach Stärke des Kantengewichts weniger oder stärker abstossen. Dabei wird je eine Version mit sichtbaren Kanten und eine Version nur mit den Vertices ohne Kanten generiert.

3.2. Klassendiagramm



3.2.1. FlowToUv

Die Klasse FlowToUv verarbeitet die CFlow-Dateien. Sie legt auch eine HashMap der Knoten im Format (IP-Adresse, KnotenNummer) an. Um mit Hilfe der Knotennummer wieder die IP-Adresse zu erhalten, wird die HashMap zusätzlich auch noch mit vertauschten Key / Values gespeichert. Da die Klasse NGraph die UV-Daten sortiert benötigt, werden diese vom Kanten-HashSet in einen Vector abgefüllt, welcher sortiert wird. Da jeder Eintrag in der Hashmap einzigartig sein muss, konnte nicht von Beginn an ein Vector verwendet werden. Um die Flows zu filtern wird die Filter-Klasse verwendet. Der Filter wird bereits beim Einlesen der einzelnen Flows angewandt, um beste Performance zu erhalten.

3.2.2. Filter

Die statische Filter-Klasse verfügt über vordefinierte Protokolle und Sets von Ports, nach denen ein Flow gefiltert werden kann. Er gibt einen booleschen Wert zurück, welcher aussagt, ob ein Flow dem angewandten Filter entspricht.

3.2.3. NGraph

NGraph berechnet aus dem UV-Vector den Nachbarschaftsgraph. Dabei kommen Knoten-Paare mehrfach vor, wenn die Knoten mehrere gleiche externe IP-Adressen besucht haben.

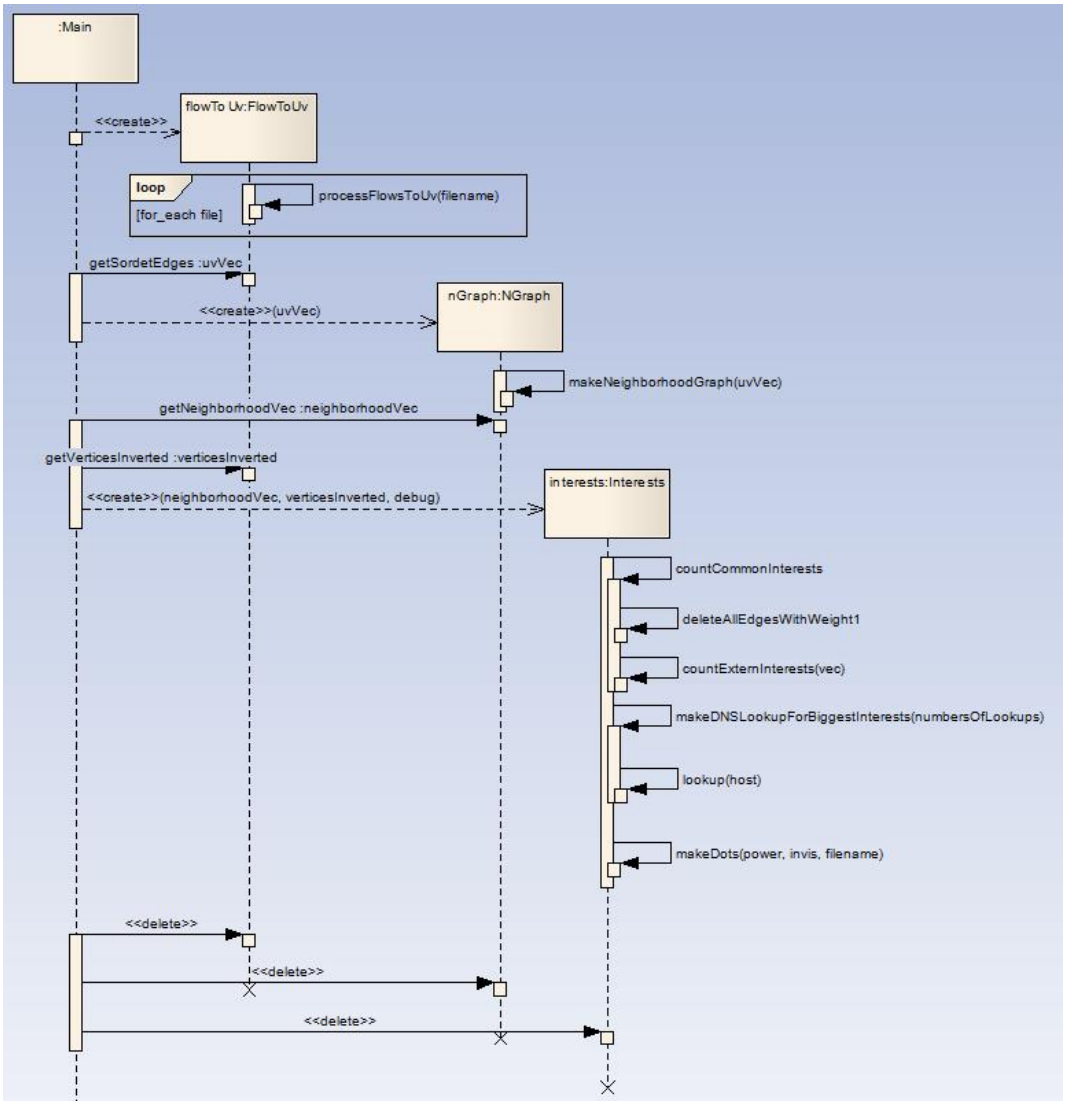
3.2.4. Interests

Die Klasse bildet aus dem Nachbarschaftsgraph die Interessen, d.h. mehrere gleiche Knoten-Paare werden zusammengefasst. Die Anzahl gemeinsamer besuchter externer Knoten stellt dabei das Kantengewicht dar. Sie erstellt das kräftebasierte Layout und schreibt es zusammen mit der Dot-Datei auf die Festplatte. Weiter macht auch das DNS-Lookup.

3.2.5. IntPair

Hilfsklasse, stellt ein Paar von Int-Werten mit Hash-Funktion und Equals-Funktion zur Verfügung, welche dort benötigt wird, wo ein IntPair (zwei Werte) als ein Wert abgespeichert wird (z.B. als Key).

3.3. Sequenzdiagramm



4. Umsetzung

4.1. Boost Graph Library

Boost verfügt über eine Graph Library [3, 4, 5] mit der Graphen repräsentiert werden können. Zu Beginn des Projektes wurde der Graph mit dieser Library gebildet. Es hat sich jedoch schnell gezeigt, dass das Kreieren und Umwandeln des Graphen sehr umständlich und komplex ist. Deshalb wurden normale Collections verwendet.

4.2. GraphViz Layout

Es wurde entschieden, die Visualisierungen der COIs mit dem GraphViz-Tool zu machen, welches auch kräftebasierte Layouts anbietet. Die zwei kräftebasierten Layouts von GraphViz sind Fdp und Neato.

Abbildung 3 zeigt ein erstes Fdp-Layout, welches mit Testdaten erstellt wurde. Die Knoten sind nicht gut verteilt. So hat z.B. Knoten 7 nur eine Kante zu Knoten 8, wird jedoch sehr nah bei den Knoten 4, 9 und 10 platziert.

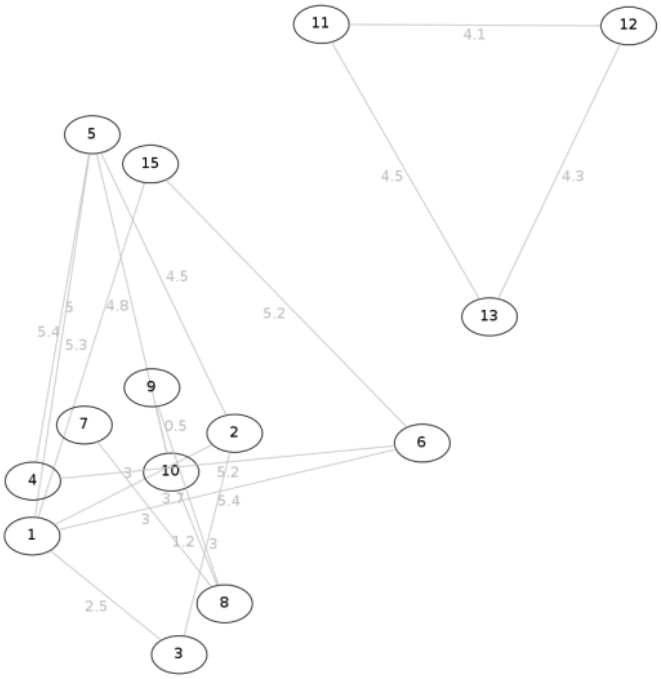


Abbildung 6: Test-Layout mit Fdp

Mit den gleichen Daten wurde ein Neato-Layout [5] erstellt. Die Knoten sind nun ohne falsche Platzierungen ihren Kanten entsprechend aufgeteilt. Es wurde deshalb entschieden, für die Visualisierung das Neato-Layout zu verwenden.

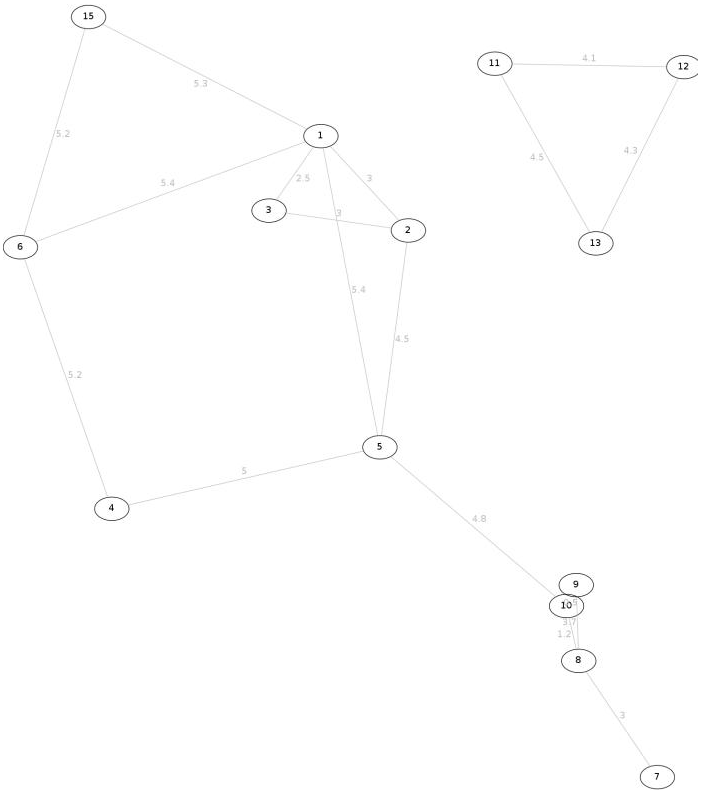


Abbildung 7: Test-Layout mit Neato

Das Neato-Layout platziert Knoten umso näher beinander, je kürzer die Kanten sind. Die Kantenlänge zwischen zwei Knoten sollte deshalb umso kleiner sein, je grösser das Kantengewicht ist. Um dies zu erreichen, wird in den Visualisierungen der Kehrwert über dem Kantengewicht gebildet. Zusätzlich wird von diesem Wert auch noch die Potenz mit Exponent 1.5 gebildet. Dies verstärkt den Distanz-Effekt: Es rückt nahe Knoten näher zusammen und weiter entferntere Knoten weiter weg.

Um die Visualisierung zu erstellen schreibt COIViz zuerst die Dot-Dateien auf die Festplatte und benutzt danach den GraphViz Neato-Systemaufruf, um die Visualisierungen als Vectordatei (.svg) zu erstellen.

4.3. Visualisierungen

Es wurden zuerst Visualisierungen mit den Daten des Analyzers gemacht, welche im Oktober 2011 aufgezeichnet wurden. Nachdem der Zugriff auf den Scylla-Cluster der ETH möglich war, wurden diese Daten visualisiert. Sie wurden Ende Juni 2011 aufgezeichnet, als das reguläre Semester bereits zu Ende war und sich deshalb kaum mehr Studenten an der HSR aufhielten, was auch in einem Rückgang des verursachten Netzwerkverkehrs sichtbar wurde.

Abbildung 5 und 6 zeigen die Visualisierung von 30 Minuten Webtraffic, welcher am 22.6.2011 von 9.00 Uhr bis 9.30 aufgezeichnet wurde. Bei Abbildung 5 wurden die

Kanten zwischen den Knoten weggelassen, damit die Knoten besser sichtbar sind. Abbildung 6 ist mit Abbildung 5 identisch, bis auf die Kanten, die hier auch dargestellt werden.

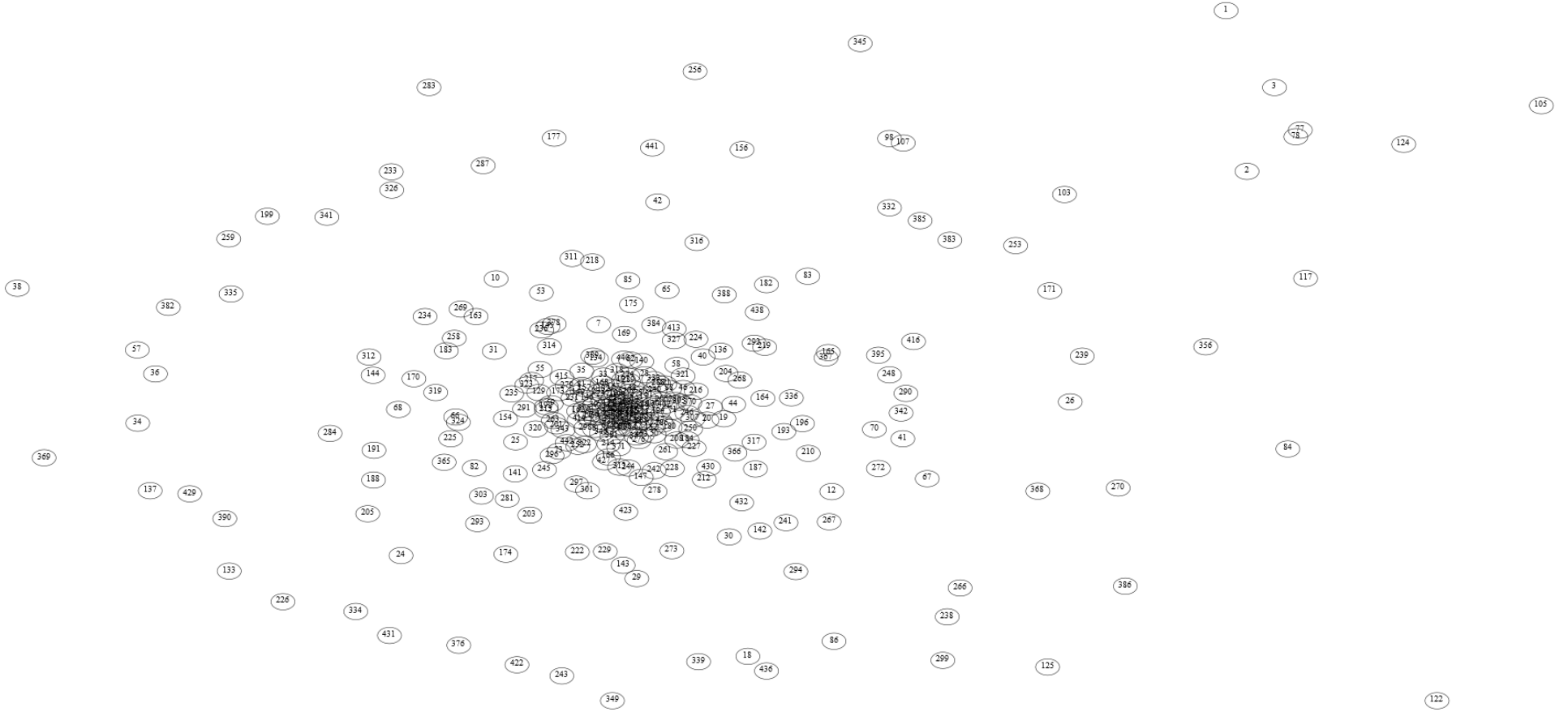


Abbildung 8: 30min Webtraffic ohne Kanten, vom 22.6.2011, 9.00Uhr

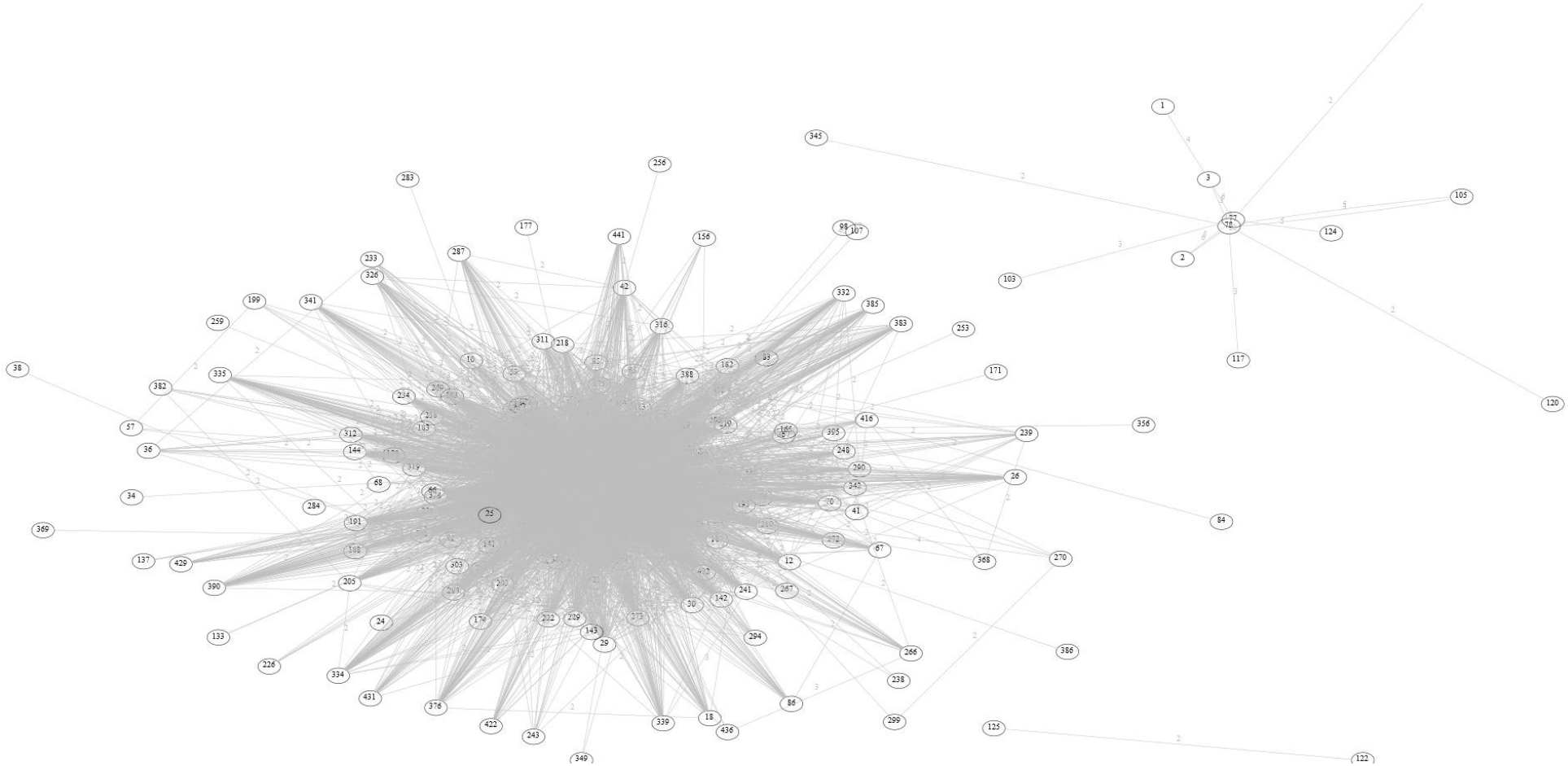


Abbildung 9: 30min Webtraffic mit Kanten, vom 22.6.2011, 9.00Uhr

4.3.1. Pseudo-COIs

Bei den Visualisierungen hat sich gezeigt, dass viele Pseudo-COIs entstehen. Beim Betrachten der Visualisierungen erwecken diese den Eindruck, richtige COIs zu sein, was sie jedoch nicht sind. Diese Pseudo-COIs entstehen durch das kräftebasierte Layout. Dieses ordnet Knoten mit hohem Kantengewicht, welche stark miteinander vernetzt sind, in der Mitte des Layouts an. Knoten mit tieferem Kantengewicht und wenig Vernetzung werden stark abgestossen und so weit weg wie möglich platziert. Solche Knoten stossen sich jedoch auch untereinander möglichst stark ab. Dies führt dazu, dass diese Knoten gleichverteilt und ringförmig um die Graphenmitte platziert werden und sich dabei näher kommen, ohne Verbindungen untereinander zu haben.

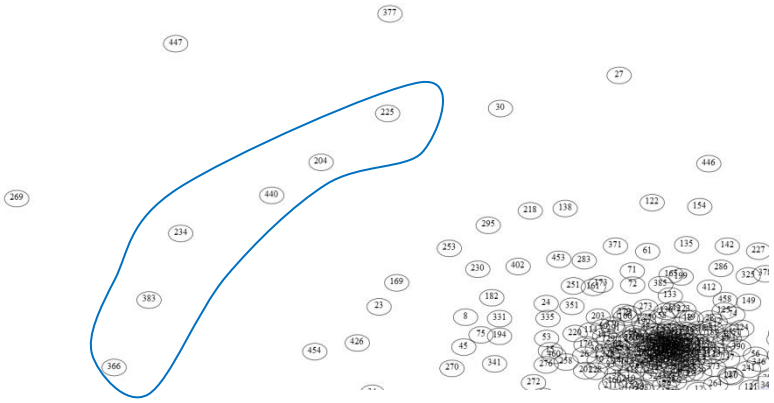


Abbildung 10: Pseudo-COI ohne Kanten

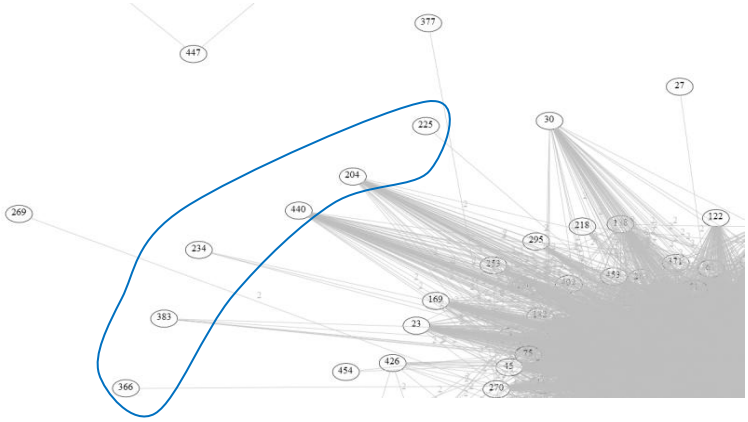


Abbildung 11: Pseudo-COI mit Kanten

4.4. DNS-Lookup

In den untenstehenden Tabellen sind die 20 von internen Knoten am häufigsten gemeinsam besuchten externen IP-Adressen am 22.6.2011, 9.00 Uhr während 30 Minuten sowie die 20 häufigsten besuchten externen IP-Adresse über den ganzen 22.6.2011 (24 Stunden). Es lässt sich nicht von allen IP-Adressen durch ein DNS-Lookup die Namen erhalten. Bei den Lookups, wo dies fehlgeschlagen ist, wurde die IP-Adresse

eingesetzt. In den Tabellen ist gut erkennbar, dass die mit grossem Abstand am häufigsten aufgerufene Webadresse Google ist, welche über mehrere verschiedene IP-Adressen erreichbar ist und durch ein Lookup nicht aufgelöst wird.

4078-mal:	209.85.146.138	209.85.146.138 //google
4022-mal:	209.85.146.100	209.85.146.100 //google
3993-mal:	209.85.146.139	209.85.146.139 //google
3982-mal:	209.85.146.113	209.85.146.113 //google
3788-mal:	209.85.146.101	209.85.146.101 //google
3496-mal:	209.85.146.102	209.85.146.102 //google
3355-mal:	74.125.79.104	ey-in-f104.1e100.net
2939-mal:	74.125.79.147	ey-in-f147.1e100.net
2771-mal:	74.125.79.99	ey-in-f99.1e100.net
1889-mal:	74.125.232.124	mil01s07-in-f28.1e100.net
1676-mal:	74.125.232.123	mil01s07-in-f27.1e100.net
1142-mal:	209.85.146.120	209.85.146.120 //google
1098-mal:	74.125.232.122	mil01s07-in-f26.1e100.net
888-mal:	74.125.232.121	mil01s07-in-f25.1e100.net
562-mal:	209.85.146.95	209.85.146.95 //google
546-mal:	2.19.79.139	2.19.79.139 //Akamai
382-mal:	80.252.91.41	80.252.91.41 //Telecity Group
348-mal:	69.63.190.14	www-11-02-ash2.facebook.com
313-mal:	8.18.45.80	ad-dc6.mediaplex.com
294-mal:	64.236.124.228	adserver.adtech.de

78610-mal:	209.85.146.102	209.85.146.102
77637-mal:	209.85.146.100	209.85.146.100
77628-mal:	209.85.146.113	209.85.146.113
75226-mal:	209.85.146.139	209.85.146.139
73442-mal:	209.85.146.138	209.85.146.138
71669-mal:	209.85.146.101	209.85.146.101
70955-mal:	74.125.79.99	ey-in-f99.1e100.net
70092-mal:	74.125.79.104	ey-in-f104.1e100.net
69685-mal:	74.125.79.147	ey-in-f147.1e100.net
55340-mal:	74.125.232.124	mil01s07-in-f28.1e100.net
53200-mal:	74.125.232.123	mil01s07-in-f27.1e100.net
41438-mal:	74.125.232.121	mil01s07-in-f25.1e100.net
37859-mal:	74.125.232.122	mil01s07-in-f26.1e100.net
35917-mal:	2.19.79.139	2.19.79.139
31889-mal:	209.85.146.95	209.85.146.95
28979-mal:	213.199.181.90	213.199.181.90
28511-mal:	209.85.146.120	209.85.146.120
26351-mal:	212.47.171.93	212.47.171.93
26019-mal:	2.19.76.20	2.19.76.20
25971-mal:	2.19.69.115	2.19.69.115

4.5. Metrik

Um den Anstieg der gemeinsamen Interessen über die Beobachtungsdauer zu untersuchen wurde eine Metrik definiert. Diese setzt sich aus der Summe aller Kantengewichte geteilt durch die Anzahl Kanten zusammen.

$$Metrik = \frac{\sum Kantengewicht}{Anzahl\ Kanten}$$

Dabei wurde die Metrik zuerst alle 10 Minuten, dann jede Stunde, 6 Stunden und 12 Stunden gemessen. Flows mit Port 53 oder Port 0 wurden nicht berücksichtigt. Auch Kanten mit einem Kantengewicht von 1 wurden herausgefiltert.

4.5.1. 10-Minuten-Dateien

Hier ist der anfängliche Anstieg der Metrik gut zu erkennen, welcher um 9.50 Uhr abflacht und um 10.10 Uhr wieder etwas zunimmt. Dieser Zeitraum entspricht an der HSR der grossen Pause, während der anscheinend weniger Netzwerkverkehr produziert wird.

10min-Dateien fortlaufend					
Datum der Aufnahme	Zeit	Dauer von Beginn (min)	Summe Kantengewicht	Anzahl Kanten	Metrik
13.10.2011 (Do)	9.10	0	0	0	0
	9.20	10	13850	6666	2.08
	9.30	20	107300	36455	2.94
	9.40	30	268360	74238	3.61
	9.50	40	442109	109894	4.02
	10.00	50	598340	144661	4.14
	10.10	60	847765	197427	4.29
	10.20	70	1109346	243390	4.56
	10.30	80	1427857	292156	4.89
	10.40	90	1826714	345669	5.28
	10.50	100	2194992	388798	5.65

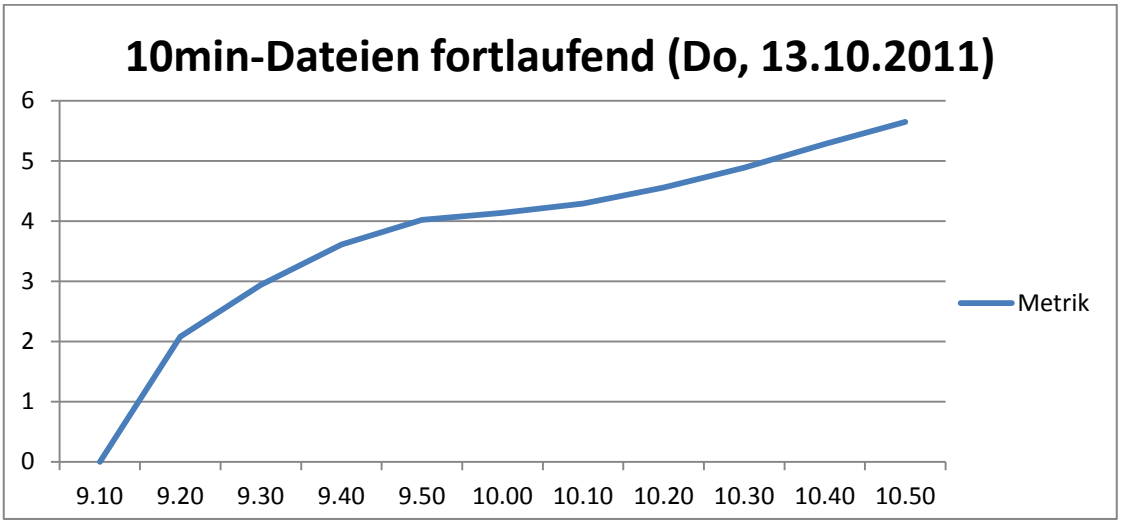


Abbildung 12: Metrik 10min-Dateien

4.5.2. 1-Stunden-Dateien

In der stündlich ausgewerteten Metrik ist vor allem die Abflachung am Nachmittag sichtbar.

1h-Dateien fortlaufend					
Datum der Aufnahme	Zeit	Dauer von Beginn (h)	Summe Kantengewicht	Anzahl Kanten	Metrik
13.10.2011 (Do)	9.10	0h	0	0	0
	10.10	1h	847729	197427	4.29
	11.10	2h	3289902	527123	6.24
	12.10	3h	6511056	836985	7.8
	13.10	4h	9420746	1035009	9.1
	14.10	5h	12160795	1209541	10.05
	15.10	6h	14490645	1335154	10.85
	16.10	7h	16276246	1454046	11.19
	17.10	8h	17149348	1503107	11.41
	18.10	9h	17951091	1557901	11.52
	19.10	10h	18658669	1605779	11.62
	20.10	11h	19420083	1660773	11.69
	21.10	12h	19963805	1685904	11.84

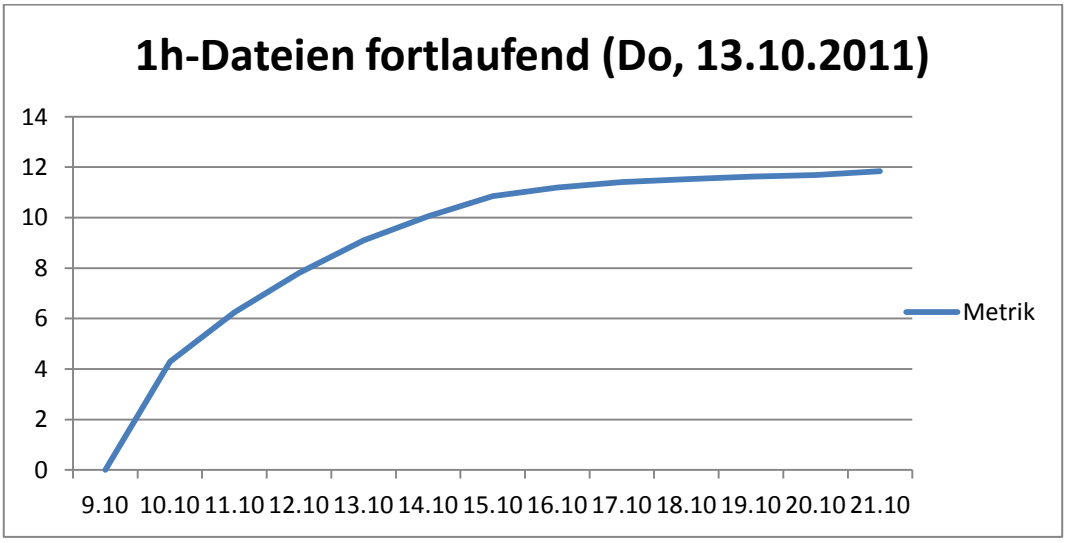


Abbildung 13: Metrik 1h-Dateien

4.5.3. 6-Stunden-Dateien

Für die Metrik im Sechsstundentakt wurde eine längere Beobachtungsdauer gewählt. Diese begann am 13.10.2011 um 9.10 Uhr und dauerte bis am 15.10.2011 15.10 Uhr. Auch hier zeigt sich ein Anstieg der Metrik am Morgen bis in den Nachmittag hinein und die Abflachung am Abend und über die Nacht. Dabei entsteht ein Treppeneffekt. Es ist auch zu erkennen, dass am Samstagmorgen der Anstieg nicht so stark ausgeprägt ist, wie an den Wochentagen.

6h-Dateien fortlaufend					
Datum der Aufnahme	Zeit	Dauer von Beginn (h)	Summe Kantengewicht	Anzahl Kanten	Metrik
13.10.2011 (Do)	9.10	0h	0	0	0
	15.10	6h	14490645	1335154	10.85
	21.10	12h	19963805	1685904	11.84
14.10.2011 (Fr)	3.10	18h	20549001	1737220	11.83
	9.10	24h	37498268	2409568	15.56
	15.10	30h	60610035	2878005	21.06
	21.10	36h	64472917	3003099	21.47
15.10.2011 (Sa)	3.10	42h	64917805	3026498	21.45
	9.10	48h	66588640	3090483	21.55
	15.10	54h	72063643	3196631	22.54

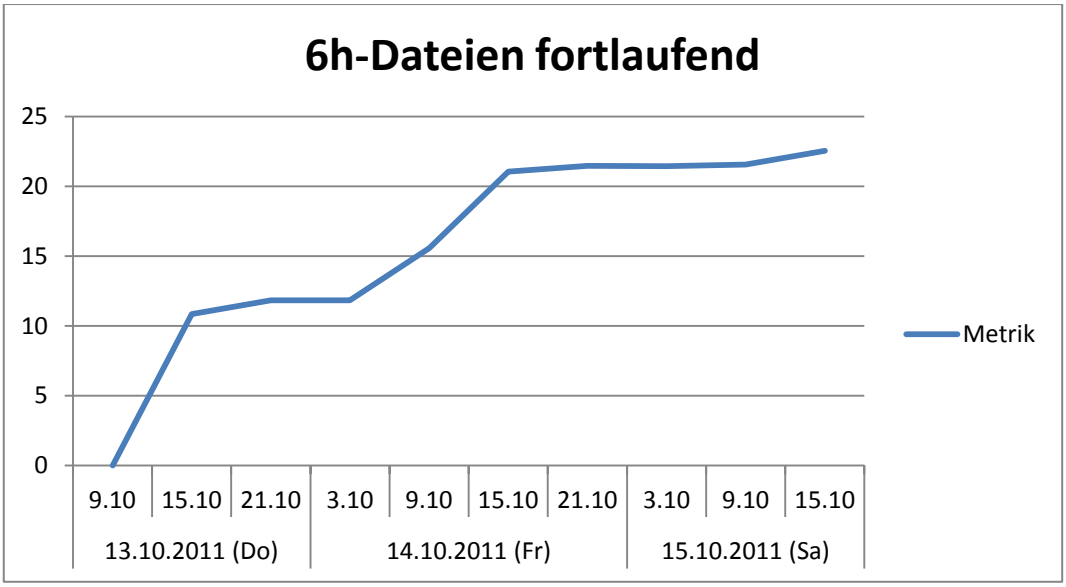


Abbildung 14: Metrik 6h-Dateien

4.5.4. 12-Stunden-Dateien

Der Treppeneffekt von der 6-Stunden Metrik geht hier grösstenteils verloren und ist nur noch andeutungsweise sichtbar, was sich durch die 12-Stunden Beobachtungdauer und dem Zeitpunkt der Metrik jeweils um 9.10 Uhr / 21.10 Uhr erklären lässt. Die Metrik steigt über das Wochenende auch tagsüber kaum an. Ein Aufwärtstrend ist erst am Montagmorgen erkennbar.

12h-Dateien fortlaufend					
Datum der Aufnahme	Zeit	Dauer von Beginn (h)	Summe Kantengewicht	Anzahl Kanten	Metrik
13.10.2011 (Do)	9.10	0h	0	0	0
	21.10	12h	19963805	1685904	11.84
14.10.2011 (Fr)	9.10	24h	37498268	2409568	15.56
	21.10	36h	64472917	3003099	21.47
15.10.2011 (Sa)	9.10	48h	66588640	3090483	21.55
	21.10	60h	74612003	3252155	22.94
16.10.2011 (So)	9.10	72h	75872882	3281868	23.12
	21.10	84h	87389349	3448230	25.34
17.10.2011 (Mo)	9.10	96h	113809905	3804353	29.92

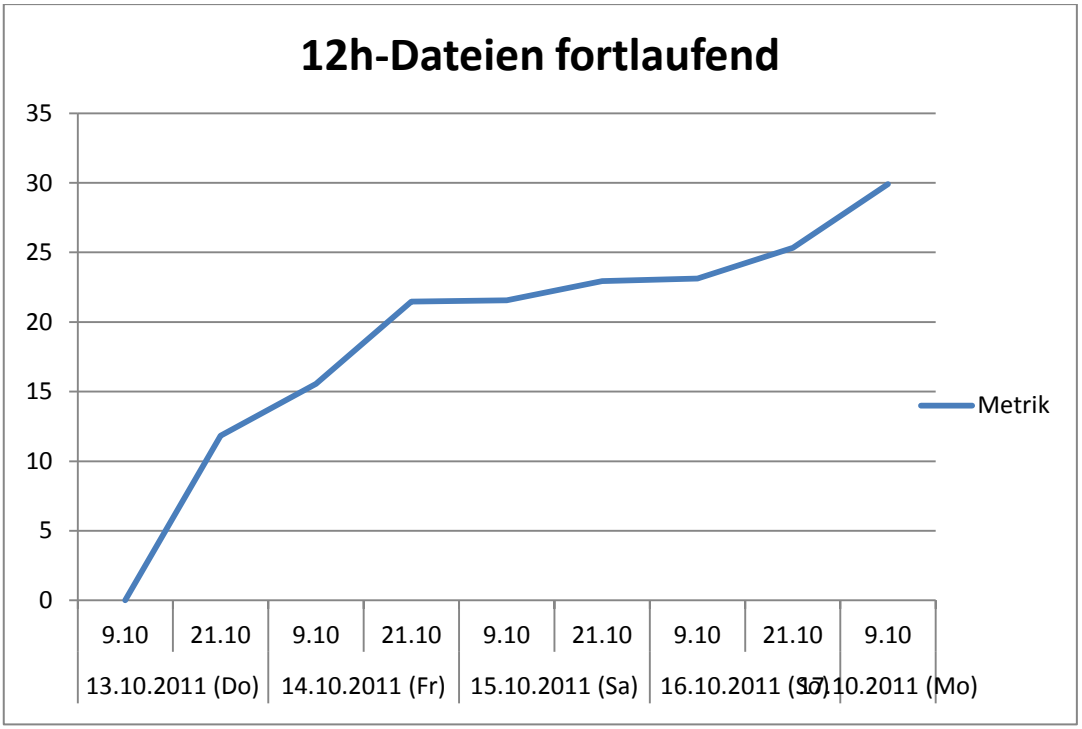


Abbildung 15: Metrik 12h-Dateien

4.6. Tests & Analyse

4.6.1. Unit Tests

Um die Grundfunktionen zu testen wurden Unit-Tests geschrieben. Dabei wird das Einlesen von CFlow-Dateien und das Filtern von Flows nicht getestet, da die CFlow-Dateien nicht von den Remote-Maschinen (Analyzer, Scylla-Cluster) entfernt werden durften.

4.6.2. Callgrind-Analyse

Callgrind[6] ist ein Profiling Tool aus dem Valgrind-Framework [7], mit dem die Aufrufhierarchie eines Programmes und seine Laufzeit analysiert werden kann, ohne dass das Programm dafür extra kompiliert werden muss. Dabei sollte beim Kompilieren jedoch die Optimierung eingeschaltet sein. Mit dem Visualisierungs-Tool KCachegrind[6] können die Profiling-Ergebnisse dargestellt werden. Dabei sind vor allem Funktionen interessant, in denen viel Zeit verbracht wird, ohne dass weite Funktionsaufrufe ausgeführt werden.

Für das Profiling von COIViz wurde die Optimierung auf höchste Stufe gesetzt. Das Programm wurde mit einer 10 Minuten langen CFlow-Datei und mit sechs 10 Minuten Dateien (= 60 Minuten) aufgerufen. Dabei zeigte sich, dass viel Zeit in der Funktion inflate aus der Library libz.so.1.2.3.3 verbracht wurde. Diese Funktion entpackt komprimierte Daten. Die Erstellung des Nachbarschaftsgraphen in der Funktion NGraph::makeNeighborhoodGraph benötigt auch viel Zeit. Dabei nimmt die Zeit für die Erstellung des Nachbarschaftsgraphen mit grösseren Datenmengen stark zu. Callgrind erfasst dabei jedoch nicht das Erstellen der GraphViz Layouts, welches der zeitintensivste Funktionsaufruf ist. Dies liegt daran, dass die Visualisierungen über einen Systemaufruf kreiert werden, welcher von Callgrind nicht erfasst wird.

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x0000000000000A60	ld-2.7.so
99.62	0.00	1	0x00000000004051E0	COIViz2
99.62	0.00	1	(below main)	libc-2.7.so
99.57	0.08	1	main	COIViz2: Main.cpp, basic_
71.72	0.46	1	FlowToUv::processFlowsToU...	COIViz2: FlowToUv.cpp, b
48.04	0.53	1	CFlowlist::read_flows()	COIViz2: flowlist.cpp, bas
47.16	0.85	104 983	CFlowlist::read_flow(boost::...	COIViz2: flowlist.cpp, istr
46.31	0.69	104 995	std::istream::read(char*, lo...	libstdc++.so.6.0.10
44.69	2.58	104 995	<cycle 1>	libstdc++.so.6.0.10
40.42	0.59	39 370	long boost::iostreams::sym...	COIViz2: symmetric.hpp,
35.39	0.02	39 744	boost::iostreams::detail::zli...	libboost_iostreams.so.1.3
35.37	33.56	39 744	inflate	libz.so.1.2.3.3
13.76	5.99	42 159	vfprintf	libc-2.7.so
11.31	0.26	30 656	inet_ntop	libc-2.7.so
11.31	0.08	30 424	util::ipV4AddressToString(u...	COIViz2: utils.cpp
10.92	0.14	30 822	sprintf	libc-2.7.so
10.78	0.26	30 822	vsprintf	libc-2.7.so
10.48	0.00	1	NGraph::NGraph(std::vecto...	COIViz2: NGraph.cpp, ha
10.48	10.44	1	NGraph::makeNeighborhoo...	COIViz2: NGraph.cpp, stl
8.36	0.12	2	Interests::makeDots(bool, s...	COIViz2: Interests.cpp, b
7.53	0.75	1	Interests::countCommonIn...	COIViz2: Interests.cpp, st
7.31	1.39	30 424	FlowToUv::insertVertexAnd...	COIViz2: FlowToUv.cpp, st
5.56	0.14	11 320	std::ostream& std::ostream...	libstdc++.so.6.0.10
5.37	0.02	11 320	std::num_put<char, std::o...	libstdc++.so.6.0.10
5.34	0.28	11 320	std::ostreambuf_iterator<c...	libstdc++.so.6.0.10
4.91	0.46	91 288	std::string::string(char con...	libstdc++.so.6.0.10
4.78	0.63	191 250	operator new(unsigned long)	libstdc++.so.6.0.10

Abbildung 16: 10min Daten

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x0000000000000A60	ld-2.7.so
99.95	0.00	1	0x00000000004051E0	COIViz2
99.95	0.00	1	(below main)	libc-2.7.so
99.94	0.04	1	main	COIViz2: Main.cpp, basic_s
55.79	0.34	6	FlowToUv::processFlowsT...	COIViz2: FlowToUv.cpp, ba
37.67	0.42	6	CFlowlist::read_flows	COIViz2: flowlist.cpp, basi
36.98	0.67	602 870	CFlowlist::read_flow	COIViz2: flowlist.cpp, istre
36.32	0.54	602 942	std::istream::read	libstdc++.so.6.0.10
35.05	2.02	602 942	<cycle 1>	libstdc++.so.6.0.10
31.72	0.46	226 084	long boost::iostreams::sy...	COIViz2: symmetric.hpp, s
27.78	0.01	228 261	boost::iostreams::detail::z...	libboost_iostreams.so.1.3
27.77	26.35	228 261	inflate	libz.so.1.2.3.3
17.77	0.00	1	NGraph::NGraph	COIViz2: NGraph.cpp, hash
17.77	17.69	1	NGraph::makeNeighborho...	COIViz2: NGraph.cpp, stl_v
13.49	2.24	1	Interests::countCommonIn...	COIViz2: Interests.cpp, stl_
13.42	5.08	296 633	vfprintf	libc-2.7.so
12.26	0.14	2	Interests::makeDots	COIViz2: Interests.cpp, bas
8.77	0.06	173 184	util::ipV4AddressToString	COIViz2: utils.cpp
8.77	0.20	174 432	inet_ntop	libc-2.7.so
8.46	0.11	175 180	sprintf	libc-2.7.so
8.35	0.20	175 180	vsprintf	libc-2.7.so
8.21	0.21	121 423	std::ostream& std::ostrea...	libstdc++.so.6.0.10
7.92	0.04	121 423	std::num_put<char, std::o...	libstdc++.so.6.0.10
7.89	0.41	121 423	std::ostreambuf_iterator<...	libstdc++.so.6.0.10
6.63	0.15	121 423	0x0000000000084F80	libstdc++.so.6.0.10
6.31	0.17	121 423	vsnprintf	libc-2.7.so
5.52	1.09	173 184	FlowToUv::insertVertexAn...	COIViz2: FlowToUv.cpp, st
4.87	0.68	1 522 799	operator new	libstdc++.so.6.0.10

Abbildung 17: 60min Daten

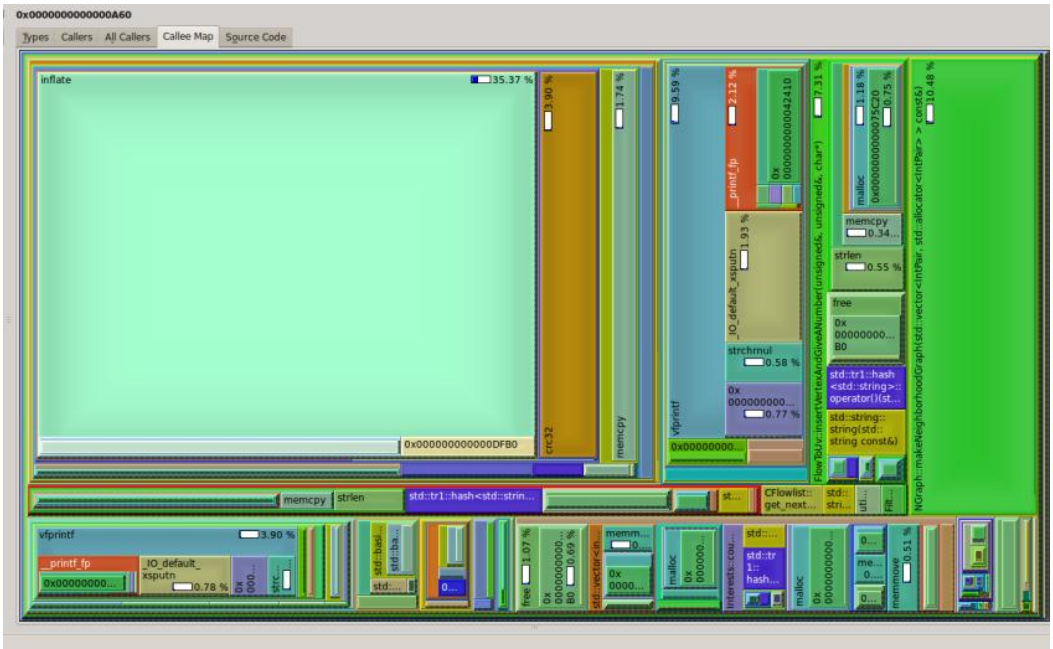


Abbildung 18: Caller Map der 10min Daten

5. Schlussfolgerung

Die Anforderungen und das Vorgehen wurden während des Projektes regelmässig den neuen Erkenntnissen angepasst. Es konnten viele der Anforderungen umgesetzt werden. Das interaktive Aufrufen der IP-Adressen wurde jedoch nicht umgesetzt.

Diese Arbeit hat jedoch gezeigt, dass der hier gewählte Ansatz für die Visualisierung von Interessensgruppen nicht funktioniert.

Dennoch ist die Visualisierung von Interessensgruppen ein äusserst interessantes Thema, welches vielleicht in einer späteren Arbeit mit einem neuen Ansatz nochmals untersucht werden kann.

Teil II

Berichtsanhang

6. Aufgabenstellung

Aufgabenstellung zur Studienarbeit HS 2011

„Visualisierung von Interessengruppen in Internet-Verkehrsdaten“

Gruppe: Marion Frei

Ausgangssituation

Endbenutzer haben entsprechend dem Computereinsatz und der Lokalisierung im Netzwerk unterschiedliche gemeinsame Interessen der Art, dass sie gemeinsame Server bzw. Dienste nutzen. So lassen sich Interessengruppen (Communities-of-Interest, COI) identifizieren, die nicht nur eine Einteilung von Endsystemen in ähnliche Gruppen erlauben, sondern auch die Erkennung von relevanten Verhaltensänderungen unterstützen.
Eine Möglichkeit solche Interessengruppen zu beobachten, stellt die Visualisierung derselben dar. Dabei stellt sich das Problem, dass sich die einzelnen Interessengruppen mehr oder weniger überlappen, das heisst in der Regel keine scharfen Abtrennungen existiert.
Die Idee besteht nun darin, durch eine geeignete Visualisierung die Interessengruppen optisch einfach erkennbar zu machen, wozu verschiedene Möglichkeiten existieren. Entsprechend besteht die Aufgabe dieser Arbeit darin, interessante Möglichkeiten zu implementieren und anhand echter Verkehrsdaten zu evaluieren. Dazu steht uns eine spezielle Infrastruktur an der HSR zur Verfügung, die uns erlaubt sämtliche Internet-Verkehrsverbindungen der HSR zu analysieren.

Aufgabe

Das Ziel dieser Arbeit besteht darin, Interessengruppen aus Internet-Verbindungsdaten zu extrahieren und zu visualisieren. Als Resultat soll ein C++ Programm unter Unix/Linux entstehen, das Internet-Verkehrsdaten einliest und visualisiert. Zudem soll es möglich sein, interaktiv einen Knoten im dargestellten Graphen zu klicken um dessen IP-Adresse zu erhalten, die dann zum Aufruf von Detail-Analyse-Tools, z.B. HAPviewer [1, 2], dienen kann.
Optional kann zusätzlich noch ein COI-Algorithmus integriert werden, der es erlaubt Interessengruppen optisch zu markieren, z.B. durch farbliche Unterscheidung oder durch Abgrenzungslinien.

Ansatz

Zur Extrahierung der Interessengruppen wird aus dem Kommunikations-Graph, der End-zu-Endsystem-Verbindungen darstellt, ein Nachbarschafts-Graph abgeleitet, in dem HSR-interne Rechner als Knoten erscheinen und die Kanten gemeinsam kontaktierte externe Rechner darstellen. Als Kanten-Annotierung wird dabei die Anzahl gemeinsamer externer Nachbarn festgehalten.
Um die optische Cluster-Bildung zu realisieren, wird sodann ein kräftebasiertes (*force-directed*) Graphen-Layout benutzt, wobei als „Anziehungskräfte“ die Kanten mit ihren Annotierungen

dienen.
Für das Graphen-Layout [3, 4] wie auch für das Einlesen und Aufbereiten von Internet-Verbindungsdaten stehen fertiger Code bzw. einfach anpassbare Code-Beispiele zur Verfügung. Ferner kann für die optionale COI-Markierung ein bestehender Algorithmus [5] genutzt werden, der als C++ Code beim Betreuer vorliegt.

Berichtsgestaltung

Der Bericht ist gemäss [6] zu gestalten.

Referenzen

[1] Glatz, E., „HAPviewer - Host Application Profile Viewer." Open Source Projekt, Verfügbar unter: http://hapviewer.sourceforge.net/ sowie http://sourceforge.net/projects/hapviewer/

[2] Glatz, E., „Visualizing Host Traffic through Graphs", 7th International Symposium on Visualization for Cyber Security (VizSec), Ottawa, Ontario, Canada, Sep. 2011.

[3] Emden R. Gansner und Stephen C. North, „Improved Force-Directed Layouts". In Graph Drawing. Lecture Notes in Computer Science, 1998, Volume 1547/1998, 364-373.

[4] Graphviz Project, „Graphviz - Graph Visualization Software". Verfügbar unter: http://www.graphviz.org/

[5] Clauset A, Newman MEJ, Moore C., „Finding community structure in very large networks". Physical Review E. 2004; 70:66111.

[6] Glatz, E., „Vorgaben zur Berichtserstellung", Ausgabe des 18. September 2011

Termine

19. Sept. 2011	Arbeitsbeginn
23. Dez. 2011	Abgabe des Berichts an den Betreuer (bis 17:00)

Weitere Termine siehe Terminangaben auf dem HSR-Web (intern).

Betreuung

Betreuer: Prof. Eduard Glatz
Während der Durchführung der Arbeit findet nach Möglichkeit regelmässig jede Woche eine Besprechung mit dem Betreuer statt. Dazu werden entsprechende Termine bei Arbeitsbeginn festgelegt.

Rapperswil, den 18. Sept. 2011

Eduard Glatz

7. Projektplan

Es wurde eine rollende Planung angewandt. In wöchentlichen Sprints wurde mit dem Betreuer jeweils das Erreichte besprochen und das weitere Vorgehen anhand dieser Ergebnisse festgelegt.

Eine erste Zeiteinschätzung wurde am Anfang der Arbeit erstellt. Der Projektplan wurde danach jeweils wöchentlich für die folgende Woche überarbeitet.

Detailplan

				Sprint 1		Sprint 2		Sprint 3		Sprint 4		Sprint 5		Sprint 6		Sprint 7		Sprint 8				Sprint 9				Sprint 10		Sprint 11			
				SW 0		SW 1		SW 2		SW 3		SW 4		SW 5		SW 6		SW 7		SW 8		SW 9		SW 10		SW 11		SW 12		SW 13		SW 14	
Woche (von - bis)				12.09	16.09	19.09	23.09	26.09	30.09	03.10	07.10	10.10	14.10	17.10	21.10	24.10	28.10	31.10	04.11	07.11	11.11	14.11	18.11	21.11	25.11	28.11	02.12	05.12	09.12	12.12	16.12	19.12	23.12
		Total																													
Arbeitspaket	**Geschätzt**	**SOLL**	**IST**	S	I	S	I	S	I	S	I	S	I	S	I	S	I	S	I	S	I	S	I	S	I	S	I	S	I	S	I
Projekt Management	**56.0**	**47.5**	**53.0**																												
Besprechungen	20	14.0	15.0			1.5	1.5	1.5	2.0	1.5	1.0	1.5	1.0	1.5	1.0	1.0	1.0	1.0	0.5	1.0	4.0			1.0	1.0			1.0	1.0	1.5	1.0		
Zeitplan	6	10.5	8.5			3.0	2.0			0.5	1.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1.0	2.0	
Version Control System	2	1.0	1.5			1.0	1.0		0.5																						
Einrichten Arbeitsumgebung	8	10.0	17.5			2.5	5.5	5.0	5.5	1.0	2.0									0.5	0.5			1.0	4.0						
Einarbeitung in Themen	10	8.0	5.5			3.0	1.5	5.0	4.0																						
Risikomanagement	10	4.0	5.0			2.0	2.0			0.5	1.0					0.5	0.5					0.5	0.5							0.5	1.0
Qualitätsmassnahmen	**26.0**	**25.0**	**36.5**																												
Review Code	10	5.5	5.0																	2.0	1.5	1.0				1.0	2.0	1.5	1.5		
Refactoring	12	15.5	25.0										2.5	2.0	3.0	2.0	4.0			2.0	1.5					2.0	3.5	4.0	5.0	3.5	4.0		1.5
Dokumente überprüfen	4	4.0	6.5																1.0			3.0	2.0		2.5			1.0	1.0			
Business Modeling	**12.0**	**15.0**	**9.5**																												
Domain Model / Klassendiagramm	6	8.0	3.5									1.0		2.0	1.5							2.0		2.0	1.0			1.0	1.0		
System Sequenz Diagramm	6	7.0	6.0											2.0	1.0							1.5		1.5	2.5			2.0	2.5		
Requirements	**5.0**	**4.5**	**6.5**																												
Anforderungsspezifikation	3	3.0	3.5									1.0	1.5													1.5	1.5			0.5	0.5
Glossary	2	1.5	3.0						0.5											0.5										1.5	2.0
Implementation	**95.0**	**81.0**	**93.5**																												
Daten einlesen	15	24.0	18.5				1.5	4.0	2.0	13.0	11.0					7.0	4.0														
Bipartiter Graph	12	6.5	13.0									4.0	5.0		3.0	1.0	3.0	1.5	2.0												
Nachbarschaftsgraph	18	14.0	20.0									6.0	6.5	2.0	5.0	1.0	2.5	5.0	6.0												
Kräftebasiertes Layout	15	3.0	3.0											1.5	2.0	1.5	1.0														
Visualisierung GraphViz	12	10.5	9.5							1.0	2.0	2.0	0.5	3.0	2.0	2.0	1.5	2.5	3.5												
Filtern	10	9.0	7.5															4.0	4.0	5.0	3.5										
Externe IP-Adresse mitführen	5	4.0	6.5																	2.0		2.0	6.5								
DNS-Lookup	5	6.0	9.5																				2.5	4.0		2.0	5.0		1.0		
Filter für HSR-Daten	3	4.0	6.0																					3.0	4.5		1.0	1.0	0.5		
Tests	**17.0**	**13.0**	**13.5**																												
Unit Tests	8	11.0	5.5									1.0		3.0		3.0	1.0	2.0	2.0			2.0	2.0								0.5
Bugfixing	5	2.0	2.5																											1.0	0.5	1.0	2.0
Profiling	4	3.5	5.5																			3.5	3.0						1.5		1.0
Dokumentation und Deployment	**36.0**	**38.0**	**40.5**																												
Projektplan	6	4.0	1.0					2.0	1.0			1.0																1.0			
Dokumente	15	34.0	27.0																	2.0	1.0	3.0		4.0	2.0	5.0	2.0			8.0	7.0	12.0	15.0
Metrik	12	3.5	7.0																	3.5	5.5		1.5								
Doxygen	3	6.0	5.5																							4.0	2.0	2.0	3.5		
Wochentotal	**247.0**	**224.0**	**253.0**	**0.0**	**0.0**	**13.0**	**15.0**	**17.5**	**15.5**	**17.5**	**18.0**	**18.0**	**17.5**	**17.5**	**19.0**	**19.5**	**19.0**	**16.5**	**19.5**	**18.5**	**18.5**	**19.0**	**18.5**	**17.0**	**18.0**	**16.0**	**17.5**	**15.0**	**17.5**	**15.0**	**17.0**	**17.0**	**22.5**

8. Projektrisikoabschätzung

Die Risikoabschätzung wurde alle zwei bis drei Wochen neu beurteilt.

ID	Risiko	Auswirkung	Massnahme	Massnahme in h	Max. Schaden in h	Wahrscheinlichkeit	Gewichteter Schaden
R01	Probleme beim Umgang mit Linux-Betriebssystem	Langes Nachschlagen / Googlen		0	10	5%	0.5
R02	Schwierigkeiten bei Programmierung von C++	Es wird mehr Zeit zum Coden benötigt	C++ Nachschlagewerke besorgen	2	25	10%	2.5h
R03	Interessensgruppen aus Graph lassen sich nicht erkennbar machen	Graph wird unübersichtlich, Interessensgruppen nicht erkennbar, Gesamtziel in Frage gestellt	Ausgangsdaten filtern um bessere Abgrenzung zu erhalten, Interessensgruppen einfärben (erfordert COI-Algorithmus), frühes Erstellen eines Prototypes	20	35	5%	1.75h
R04	Verarbeitung der Daten benötigt viel Zeit	Programm hat lange Laufzeit	Performante Collections nutzen (Hashsets / -maps), Programm mit Analyse-Tools wie z.B. Valgrind analysieren um Schwachstellen zu finden	15	20	20%	4h
R05	Schwierigkeiten beim Kompilieren von COIViz auf ETH-Cluster, da nur beschränkt Rechte vorhanden sind	Nach Lösung googlen	Zusätzliche Abhängigkeiten lokal installieren, LD_LIBRARY_PATH setzen	3	15	2%	0.3h

9. Build Anleitung

9.1. Abhängigkeiten

Siehe Kapitel 2.1.5

Um COIViz auszuführen, muss die COIVizLib vorhanden sein.

9.2. Eclipse-Projekt

COIViz besteht aus insgesamt 3 Eclipse-Projekten. Diese können in Eclipse importiert und dort kompiliert werden.

9.2.1. COIVizLib

Die Library, welche die von Prof. Eduard Glatz zur Verfügung gestellten Klassen enthält.

9.2.2. COIViz

Das eigentliche Programm. Enthält die Source-Dateien für COIViz, die CUTE-Tests sowie das Doxyfile um die Doxygen-Dokumentation des Source-Codes zu erstellen. Die Code Dokumentation liegt bereits generiert im HTML und Latex Format vor.

Damit die Tests kompilieren, muss zuerst die Main-Funktion in der Datei Main.cpp auskommentiert und die Main-Funktion in der Datei Test.cpp einkommentiert werden.

9.2.3. HSRFlowFilter

Das Projekt HSRFlowFilter filtert aus CFlow-Dateien alle Flows, welche als Start- oder Zieladresse die HSR-Subnetzmaske enthalten und speichert diese HSR-Flows in eine separate Datei.

10. Werkzeuge / Tools

- COIViz wurde mit der Eclipse IDE for C/C++ Developer [9] entwickelt.
- Die Tests wurden mit dem CUTE Eclipse Plugin [10] geschrieben.
- Für die Versionskontrolle wurde das SVN Eclipse Plugin und Tortoise SVN [11]verwendet.
- Das Profiling wurde mit Valgrind [8] und KCachegrind [7] erstellt.
- Für den Datenaustausch zwischen dem lokalen Rechner und den Host-Rechnern (Analyzer, Scylla-Cluster) wurde Gftp verwendet[11].

11. Sitzungsprotokolle

11.1. Woche 1

11.1.1. Datum und Ort

Donnerstag, 22.09.2011, HSR

11.1.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.1.3. Inhalt

- "Kick-off", erste Besprechung der Studienarbeit
- Grober Ablauf besprochen
- GUI mit GTKMM
- Svg-Datei für Graphendarstellung vorteilhaft

11.1.4. Ziele für folgende Woche

- HAPviewer zum Laufen bringen
- In Boost Graph Library einarbeiten
- Zeitplan erstellen
- Versionsmanagement einrichten

11.2. Woche 2

11.2.1. Datum und Ort

Donnerstag, 29.09.2011, HSR

11.2.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.2.3. Inhalt

- Name des Projekts festgelegt: COIviz
- Besprechung des Fortschritts von letzter Woche
- Ipf-Datei für Testzwecke erhalten
- Cflow2ipfix.cpp erhalten, zeigt wie ipf-Datei eingelesen werden kann

11.2.4. Ziele für folgende Woche

- Erste Definition von Meilensteinen
- Ipf-Datei einlesen und in eine u,v-Graphenbeschreibung umwandeln
- u,v-Graphenbeschreibung in DOT-Format bringen

11.3. Woche 3

11.3.1. Datum und Ort

Donnerstag, 06.10.2011, HSR

11.3.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.3.3. Inhalt

- Besprechung des Fortschritts von letzter Woche
- IP-Adresse bitweise verknüpfen um Maske zu erhalten

11.3.4. Ziele für folgende Woche

- Anforderungsspezifikation
- Nachbarschaftsgraph visualisiert mit kräftebasierten Layout

11.4. Woche 4

11.4.1. Datum und Ort

Donnerstag, 13.10.2011, HSR

11.4.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.4.3. Inhalt

- Anforderungsspezifikation besprochen
- Nachbarschaftsgraph mit kräftebasierten Layout nicht implementiert wegen Krankheit
- Erste Visualisierungen besprochen: Ergebnisse waren nicht besonders aussagekräftig. Um verschiedene kräftebasierte Layouts zu testen. Am Anfang keine echten Daten sondern Testdaten benutzen.
- Möglichkeiten für Kantengewicht besprochen. Z.B. alle 1-er Verbindungen weglassen, Gewicht mit Potenzen verstärken, …

11.4.4. Ziele für folgende Woche

- Steilheit im Kantengewicht experimentell erforschen
- Graph mit und ohne Kanten
- Nachbarschaftsgraph visualisiert mit kräftebasierten Layout

11.5. Woche 5

11.5.1. Datum und Ort

Donnerstag, 20.10.2011, HSR

11.5.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.5.3. Inhalt

- Vertraulichkeitsvereinbarung unterschrieben
- Zugang zu flow-Daten erhalten
- Kräftebasiertes Layout von N-Graph aus Testdaten besprochen
- Mögliche Alternativen / Ergänzungen zu Forde-directed Layout besprochen
  - Frequent Itemset Mining Cluster
  - Manuell erstelltes Layout

11.5.4. Ziele für folgende Woche

- Nachbarschaftsgraph mit echten Daten
- Evtl. Traffic Dispersion Graph aus bipartitem Graph
- Evtl. neue Ideen mit Frequent Itemset Mining Clustern

11.6. Woche 6

11.6.1. Datum und Ort

Donnerstag, 27.10.2011, HSR

11.6.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.6.3. Inhalt

- Erste Ergebnisse mit echten Daten besprochen, Intressensgruppen sind erkennbar
- Verschiedene Filter-Methoden besprochen
  - Auf Biflows konzentrieren
  - Filtern nach FTP / TCP & UDP / ICMP / Web / E-Mail / …
  - Port 0 herausfiltern, da dies fragmentierte Daten sind

11.6.4. Ziele für folgende Woche

- Aussagekräftige Graphen
- Ausreichende Performance

11.7. Woche 7

11.7.1. Datum und Ort

Donnerstag, 03.11.2011, HSR

11.7.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.7.3. Inhalt

- Gefilterte Daten besprochen, Cluster besser erkennbar
- Client-Applikation rückt in den Hintergrund, Konzentration auf die Aussage der Layouts
- Konzentration auf die Aussagen der Layouts
  - Client-Applikation rückt in den Hintergrund
- Die Daten der HSR sind anonymisiert, auch die externen Adressen, dass lässt leider keine Rückschlüsse über die gemeinsamen Interessen zu
  - Alternative: Zugriff auf ETH-Daten, welche nicht anonymisiert werden
  - Der Verantwortliche der HSR wird mit Marion Frei in Kontakt treten

11.7.4. Ziele für folgende Woche

- Filter weiterentwickeln
- Profiling mit Valgrind
- SSH-Keys erzeugen

11.8. Woche 8

11.8.1. Datum und Ort

Donnerstag, 10.11.2011, HSR

11.8.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.8.3. Inhalt

- Wie kann man aussagekräftige Metrik über gemeinsame Interessen definieren?
  - Summe Kantengewicht / Anzahl Kanten
- Profiling angeschaut → Optimierung sollte eingeschaltet sein
- IP-Adresse der grössten gemeinsamen Interessen herausfinden
- Heute Termin bei Brian Trammell, ETH, für den Zugriff auf die Flow-Daten der ETH

11.8.4. Ziele für folgende Woche

- Weiter wie bisher

11.9. Woche 10

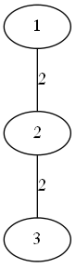
11.9.1. Datum und Ort

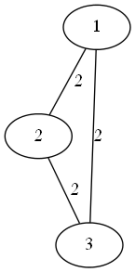
Donnerstag, 24.11.2011, HSR

11.9.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.9.3. Inhalt

- Zugriff auf ETH-Daten erhalten
- Wo Grenze für Interessensgruppe setzen?
  - 

Hier haben Knoten 1 und 3 gemeinsame Interessen mit 2, aber nicht untereinander
  - 

Richtige Interessensgruppe, alle haben Verbindungen zueinander
- Um ETH-Daten nach HSR-Daten zu filtern, wird die alte Flowlist Klasse durch die aktuelle Version ersetzt, da diese neben dem Lesen auch das Schreiben von Flows unterstützt.

11.9.4. Ziele für folgende Woche

- Anforderungsanalyse überarbeiten
- Klassendiagramm
- Sequenzdiagramm
- Code mit Doxygen dokumentieren

11.10. Woche 12

11.10.1. Datum und Ort

Donnerstag, 08.12.2011, HSR

11.10.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.10.3. Inhalt

- Abschluss der Arbeit

11.10.4. Ziele für folgende Woche

- Erste Version der Dokumentation bis Mittwoch Mittags, 14.12.2011, abgeben

11.11. Woche 13

11.11.1. Datum und Ort

Donnerstag,15.12.2011, HSR

11.11.2. Anwesende

- Prof. Eduard Glatz
- Marion Frei

11.11.3. Inhalt

- Letze Besprechungen zur Abgabe
- Dokumentstruktur der Abgabe-Doku

11.11.4. Ziele für folgende Woche

- Abgabe

12. Rückblick

Das Projekt startete leider nicht unter den besten Voraussetzungen. Kurz vor Beginn der Arbeit stellte sich heraus, dass mein Teampartner, mit dem ich diese Studienarbeit bearbeiten sollte, leider nicht die nötigen Anforderungen für das Projekt erfüllen würde und ich sie alleine bearbeiten muss. Dieser Herausforderung habe ich mich gerne gestellt und dabei auch viel über Selbstmanagement gelernt. Dabei merkte ich jedoch schnell, dass mir vor allem das gemeinsame Arbeiten und die Diskussionen und Besprechungen im Team fehlten.

Dies war mein erstes Projekt mit experimentellem Hintergrund. Es war spannend, an einem Projekt zu arbeiten, bei dem der Ausgang nicht klar sein würde und nicht von Anfang an klar war, welche Ergebnisse dabei herauskommen würden. Natürlich ist es sehr bedauerlich, dass der gewählte Ansatz in eine Sackgasse geführt hat. Auch die rollende Planung mit ständiger Anpassung der Planung war neu für mich.

Trotzdem war das Projekt sehr lehrreich und interessant. So durfte ich mich in die C++-Programmierung einarbeiten, in der ich bisher nur wenig Erfahrung sammeln konnte. Auch beim Umgang mit dem Linux-Betriebssystem konnte ich viel dazulernen. Mein Betreuer, Prof. Eduard Glatz, hat mich in allen Belangen mit seinem breiten Fachwissen dabei immer sehr gut unterstützt.

13. Glossar

Biflow	Bidirectionaler Flow Siehe auch Flow
Bipartiter Graph	Graph, der Kanten zwischen zwei Mengen aufweist. Innerhalb der Mengen gibt es keine Kanten
CFlow-Datei	Ansammlung von Netzwerkverkehrsdaten, in denen nur die Headerinformationen (Source-IP, Destination-IP, Protokoll, …) gespeichert sind Siehe auch Flow
COI	Community of Interests, gemeinsame Interessen einer Benutzergruppe
DOT-Format	Dateiendung der Graphen-Beschreibungssprache
Flow	Einzelne Netzwerkverbindung Siehe auch CFlow-Datei
Kräftebasiertes Layout	Durch die Kanten wirken auf den einzelnen Knoten des Graphen Kräfte, welche die Position des Knoten festlegen
Ipf-Datei	Anderes Dateiformat als CFlow-Datei Siehe auch CFlow-Datei
Nachbarschaftsgraph (N-Graph)	Zwei Knoten werden durch eine Kante verbunden, falls sie gleiche Interessen (Besuch von externen IP-Adressen) haben
Svg-Datei	Scalable Vector Graphic, eine skalierbare Vektorgrafik
UV Form	Knotennummern-Paar, welches eine Kante symbolisiert wobei U < V sein muss.

14. Abbildungsverzeichnis

15. Referenzen

[1] http://www.graphviz.org/Documentation.php
letzer Zugriff am 11.12.2011

[2] http://4webmaster.de/wiki/Graphviz-Tutorial
letzer Zugriff am 22.12.2011

[3] http://drdobbs.com/184402049?pgno=1
letzer Zugriff am 22.12.2011

[4] http://www.boost.org/doc/libs/1_48_0/libs/graph/doc/index.html
letzer Zugriff am 22.12.2011

[5] Siek, Lee, Lumsdaine, "The Boost Graph Library: User Guide and Reference Manual", 2002
letzer Zugriff am 22.12.2011

[6] http://www.graphviz.org/pdf/neatoguide.pdf
letzer Zugriff am 22.12.2011

[7] http://kcachegrind.sourceforge.net/html/Home.html
letzer Zugriff am 13.12.2011

[8] http://www.cprogramming.com/debugging/valgrind.html
letzer Zugriff am 13.12.2011

[9] http://www.eclipse.org/downloads/
Letzer Zugriff am 18.10.2011

[10] http://cute-test.com/
letzer Zugriff am 11.12.2011

[11] http://tortoisesvn.tigris.org/
letzer Zugriff am 4.11.2011

[12] http://www.gftp.org/
letzer Zugriff am 23.12.2011