INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Charakterisierung von Netzwerkverkehr

Arsham Sabbaghi Asl

Bachelorarbeit

Beginn der Arbeit:	03. Juni 2014
Abgabe der Arbeit:	03. September 2014
Gutachter:	Prof. Dr. Stefan Conrad
	Jun.-Prof. Dr. Dorothea Baumeister

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 03. September 2014

_____

Arsham Sabbaghi Asl

Zusammenfassung

In dieser Arbeit wird eine Charakterisierung von Netzwerkverkehr anhand von zwei Mitschnitten vorgenommen. Der erste betrachtete Mitschnitt ist am 17.09.2012 auf dem Webserver der Heinrich-Heine-Universität Düsseldorf entstanden; der zweite am 18.07.2012 auf dem Server der Fachschaft Informatik. Um verschiedene komplexe Analysen der Mitschnitte zu ermöglichen, wurde im Rahmen dieser Bachelorarbeit ein Programm entwickelt, welches Mitschnitte im pcap-Format einlesen und in eine Datenbank überführen kann.

Bevor jedoch die Implementierung des Programms besprochen werden kann, werden die verschiedenen Schichten der TCP/IP-Protokollfamilie umrissen und die für diese Arbeit relevanten Protokolle kurz erklärt.

In der nachfolgenden Evaluation werden dann einige Eigenschaften herausgearbeitet, die sich für die Netzwerküberwachung eignen. Zu diesen Eigenschaften zählen unter anderem die Anteile der verschiedenen Transportprotokolle, sowie die Anzahl der benutzten TCP-Ports pro Quell-IP. Beide Merkmale können dazu verwendet werden, Angriffe (insbesondere DoS- und DDoS-Angriffe) zu erkennen. In dieser Evaluation wird auch deutlich, dass sich viele Eigenschaften beider Mitschnitte sehr ähneln, obwohl diese sehr unterschiedlich sind.

Danksagung

An dieser Stelle möchte ich mich bei den Personen bedanken, die mich im Verlauf dieser Bachelorarbeit unterstützt haben.

An erster Stelle sei hier mein Betreuer Michael Singhof genannt, der immer schnell auf meine E-Mails geantwortet hat und jederzeit einen kurzfristigen Termin für mich gefunden hat.

Weiter möchte ich meinen Korrekturlesern Dorian Eikenberg, Philipp Rehs und Laura Schwabedissen danken, die viel Zeit mit der (un)fertigen Arbeit verbracht haben.

Nicht zuletzt möchte ich mich bei meiner Familie bedanken, die mich während meines gesamten Studiums unterstützt hat und mit der ich in dieser Phase nicht so viel Zeit verbringen konnte, wie ich gerne hätte.

Sehr wohl zuletzt möchte ich meine Freunde erwähnen, die es gerade während der Endphase der Arbeit schweren Herzens verkraftet haben, die eine oder andere Feierlichkeit ohne mich zu zelebrieren.

Inhaltsverzeichnis

1 Einleitung

Server verschiedenster Größe können unterschiedliche Dienste zur Verfügung stellen und somit sehr heterogenen Netzwerkverkehr produzieren. Eine Charakterisierung von Netzwerkverkehr und im Zuge dessen eine Analyse der Eigenschaften wäre also hilfreich, um herauszufinden, welche Attribute sich auch bei sehr unterschiedlichen Servern ähneln und welche von ihnen viele verschiedene Ausprägungen besitzen. Mit den Ergebnissen dieser Analysen kann man Aussagen darüber treffen, welche Eigenschaften sich für die Überwachung des Verkehrs eignen und welche eine zu große Varianz besitzen, um vergleichbar zu sein.

Hierfür benötigt man Mitschnitte des Verkehrs. Diese Mitschnitte enthalten im Regelfall alle Daten, die in einem bestimmten Zeitraum über eine Netzwerkschnittstelle geflossen sind. Demnach können sie beliebig groß werden, was die Analyse der vorliegenden Daten erheblich erschwert.

Hier kommen Datenbanksysteme ins Spiel. Sie sind dazu in der Lage, selbst auf großen Datenmengen, verhältnismäßig schnelle Auswertungen zu liefern.

Zuallererst muss jedoch der zu analysierende Verkehr in solch eine Datenbank geschrieben werden. Deswegen ist im Laufe dieser Arbeit ein Programm geschrieben worden, welches den Verkehr eines vorhandenen Mitschnitts parst und diesen in eine Datenbank überführt. Dieser Mitschnitt muss lediglich im *pcap*[1]-Format vorliegen und als Kapselung *DLT_EN10MB* (Ethernet), *DLT_LINUX_SLL* (Linux cooked-mode capture) oder *DLT_RAW* (Raw IP) benutzen[2]. Geparst wird mit dem Python-Modul *dpkt*[3].

Um tiefergehende Auswertungen zu ermöglichen, ist das Programm ebenfalls dazu in der Lage, für Pakete die Zugehörigkeit zu einer Verbindung auszurechnen. Dies erlaubt die Analyse von komplexeren Eigenschaften wie Verbindungslänge und Datenfluss pro Verbindung.

In Kapitel 2 werden zunächst einige wichtige Grundlagen der Funktionsweise von Rechnernetzen vermittelt, deren Kenntnisse für die Methoden dieser Arbeit wichtig sind. Im nächsten Teil der Arbeit (Kapitel 3) wird die Implementierung des Parsers erklärt und das Schema der Datenbank dargestellt. Hiernach folgt im Abschnitt 4.1 eine Beschreibung der Mitschnitte sowie in 4.2 eine tiefergehende Analyse selbiger. Nun werden die Ergebnisse dieser Analysen in den Zusammenhang der Netzwerküberwachung gebracht und ein Fazit daraus gezogen (Kapitel 5).

[1] weitere Informationen unter `www.tcpdump.org/`

[2] Link-Layer Header Types, nachzulesen unter `http://www.tcpdump.org/linktypes.html`

[3] zu finden unter `https://code.google.com/p/dpkt/`

2 Grundlagen

Netzwerkverkehr besteht aus Datenpaketen, die zwischen Systemen ausgetauscht werden. In der Regel sind sie an ein System adressiert; sie können aber zum Beispiel mittels *Broadcast*[4] auch an mehrere Netzwerkschnittstellen gerichtet sein. Wenn man den Netzwerkverkehr mitschneiden möchte, muss man also die ankommenden und abgehenden Pakete an der Schnittstelle abhören. An dieser Stelle ist *pcap* nützlich.

2.1 Lesen von Netzwerkverkehr

2.1.1 Was ist *pcap*?

pcap (von engl. **p**acket **cap**ture) ist eine Programmierschnittstelle zur Netzwerkanalyse. Sie wird zum Abgreifen von Netzwerkpaketen unmittelbar an der Netzwerkschnittstelle benutzt. Unter unixoiden Systemen liegt pcap als *libpcap* vor; bei Windows wird *WinPcap* benutzt.

Bekannte freie Programme, die libpcap benutzen, sind zum Beispiel *Wireshark*[5] und *nmap*[6].

2.1.2 Was ist *dpkt*?

dpkt ist ein Python-Modul, welches ermöglicht, schnell und bequem Pakete zu erstellen beziehungsweise zu parsen. Es kann – wie in dieser Arbeit – dazu verwendet werden, pcap-Dateien auszulesen. Außerdem ist es dazu in der Lage, Informationen der höheren Schichten der TCP/IP-Protokoll-Familie zu dekodieren.

2.2 Was ist die TCP/IP-Protokoll-Familie?

Die TCP/IP-Protokoll-Familie (auch Internetprotokollfamilie) ist eine Sammlung von Netzwerkprotokollen, auf der das Internet basiert. Sie ist wie folgt aufgeteilt:

- Die Internetschicht ist für das *Routing* zuständig, was der Navigation durch die Internet-Infrastruktur entspricht.
- Die Transportschicht sorgt für eine Ende-zu-Ende-Verbindung.
- Zur Anwendungsschicht gehören alle Protokolle, die von Anwendungsprogrammen genutzt werden.

[4] siehe [Mog84]
[5] weitere Informationen unter `https://www.wireshark.org/`
[6] weitere Informationen unter `http://nmap.org/`

Die Internetschicht ist hierbei die tiefste und die Anwendungsschicht die höchste Schicht. Pakete von Protokollen höherer Schichten sind in Pakete von Protokollen tieferer Schichten eingerahmt. Unterhalb der Internetschicht existiert noch die Netzzugangsschicht, die für die Kommunikation zwischen zwei Netzwerkschnittstellen sorgt; diese beinhaltet jedoch keine Protokolle der TCP/IP-Protokoll-Familie. Das TCP/IP-Referenzmodell ist in Tabelle 1 angegeben.

Besonders hervorzuheben sind hierbei *IPv4*, *ICMP*, *TCP* und *UDP*, da diese die für die vorliegende Arbeit relevanten Protokolle sind. Sie werden im Folgenden kurz erklärt.

TCP/IP-Schicht	Beispielprotokolle
Anwendung	HTTP, SMTP, FTP
Transport	TCP, UDP
Internet	IPv4, IPv6, ICMP
Netzzugang	Ethernet

Tabelle 1: TCP/IP-Referenzmodell

2.3 Funktionsweise von *IPv4*

IPv4[7] ist eines der wichtigsten Protokolle im Internet; es macht Netzwerkschnittstellen adressierbar. Hierbei gibt es eine Unterscheidung zwischen globalen IP-Adressen, die weltweit eindeutig sein müssen und lokalen IP-Adressen, die nur innerhalb eines privaten Netzwerks gültig sind. Jede IP-Adresse besteht aus 32 Bit, was nur 4.294.967.296 eindeutige Adressen ermöglicht, welche den weltweiten Bedarf nicht mehr abdecken[8]. Aus diesem Grund wurde das Protokoll *IPv6* eingeführt, dessen IP-Adressen aus 128 Bit bestehen und somit ungefähr $3.4 \cdot 10^{38}$ Adressen ermöglichen.

IP-Pakete besitzen einen Header, in dem Informationen wie die Versionsnummer des verwendeten IP-Protokolls sowie Quell- und Ziel-Adresse enthalten sind. Ein ebenfalls wichtiges Feld im Header ist *Protocol*. In diesem Feld steht die Protokollnummer des Protokolls der nächsthöheren Schicht[9]. Eine *1* steht zum Beispiel für *ICMP*, eine *6* für *TCP* und eine *17* für *UDP*.

ICMP[10] ist das Kontrollprotokoll für IPv4. ICMP-Pakete werden unter anderem von Routern verschickt, wenn das Ziel nicht erreichbar ist oder die Gültigkeitsdauer des Paketes überschritten wurde. Für IPv6 existiert die Alternative *ICMPv6*, welche ähnlich funktioniert. Der Header von ICMP ist recht einfach gehalten, da er nur aus den Feldern *Typ* und *Code* besteht, sowie eine Prüfsumme enthält.

[7] Internet Protocol Version 4, siehe [Pos81a]
[8] siehe `http://www.ripe.net/internet-coordination/ipv4-exhaustion`
[9] ASSIGNED NUMBERS, siehe[RP94]
[10] Internet Control Message Protocol, siehe [Pos81b]

2.4 Funktionsweise von *TCP* und *UDP*

TCP[11] und *UDP*[12] sind die wichtigsten Protokolle der Transportschicht. Sie sind dafür zuständig, Pakete, die mit IP dem richtigen System übermittelt wurden, an die richtige Anwendung im System weiterzuleiten. Dies wird anhand von Portnummern realisiert. Jedes TCP- und UDP-Paket besitzt einen Quell- und einen Ziel-*Port*. Der Quell-Port ist notwendig, um Antworten auf Pakete erhalten zu können. Ports von 0-1023 sind von der *Internet Assigned Numbers Authority*[13] standardisierte Ports.

TCP und UDP lösen diese Aufgabe jedoch unterschiedlich. TCP ist ein verbindungsorientiertes Protokoll und garantiert eine zuverlässige Datenübertragung inklusive Reihenfolgeerhaltung der Pakete. UDP wurde zeitlich nach TCP entwickelt und wurde für Anwendungen konzipiert, die echtzeitkritisch sind. Dies ist zum Beispiel bei Sprachkommunikation der Fall. Hier ist es nicht besonders wichtig, dass jedes Paket korrekt ankommt, was erheblich zu Verzögerungen führen würde. Stattdessen sollen die Pakete möglichst in Echtzeit übertragen werden. Daher ist es kaum verwunderlich, dass der TCP-Header wesentlich komplexer ist als der UDP-Header. Er besitzt zusätzlich zu Quell- und Ziel-Port eine Sequenz- und eine ACK-Nummer(von engl. acknowledgment), die den korrekten Erhalt der Pakete sicherstellen, sowie eine Reihe von Flags.

[11] Transmission Control Protocol, siehe [Pos81c]
[12] User Datagram Protocol, siehe [Pos80]
[13] weitere Informationen unter `http://www.iana.org/`

3 Implementierung

3.1 Parser

Dieses Kapitel beschreibt die Implementierung des pcap-Parsers, auf dem die Ergebnisse dieser Arbeit basieren. Mit diesem ist es möglich, Netzwerkverkehr, der im pcap-Format vorliegt, in eine Datenbank zu übertragen. Das Programm ist zum Zeitpunkt der Abgabe dazu in der Lage, Dateien zu parsen, die als Kapselung *DLT_EN10MB* (Ethernet), *DLT_LINUX_SLL* (Linux cooked-mode capture) oder *DLT_RAW* (Raw IP) benutzen.

Der Parser erstellt beim ersten Aufrufen zunächst eine Datenbank und baut eine Verbindung auf diese auf. Als Datenbanksystem wurde hierbei *SQLite* als relationales Datenbanksystem gewählt, da SQLite keine weitere Software braucht, die im Hintergrund läuft und daher sehr komfortabel zu bedienen ist. Des Weiteren ist die Datenbank auf eine Datei begrenzt, was die Benutzung weiter vereinfacht.

Es wird für IP, TCP, UDP und ICMP jeweils eine Tabelle angelegt. In der Tabelle IP wird nun für jedes IP-Paket ein Eintrag mit relevanten Daten wie Zeitstempel, Quell- und Ziel-IP-Adresse, sowie Protokollnummer des IP-Headers hinzugefügt. Je nach Protokollnummer wird ebenfalls eine neue Zeile in TCP, UDP oder ICMP angelegt, wobei diese Zeile die gleiche Paket-ID (kurz *PID*) besitzt, wie das zugehörige IP-Paket. Diese Zeilen enthalten dann weitere protokoll-spezifische Informationen, wie Portnummern und Flags.

Da in dieser Arbeit nur IP-Pakete betrachtet werden, werden Protokolle mit anderen Aufgaben – wie zum Beispiel *ARP*[14] – ignoriert und Pakete dieser Protokolle nicht in die Datenbank eingetragen. Es wird allerdings gezählt, wie viele solcher Pakete in der pcap-Datei enthalten sind und es wird gegebenenfalls die Protokollnummer dieses Protokolls ausgegeben.

Das Programm bietet zusätzlich die Möglichkeit, für TCP-Pakete auszurechnen, zu welcher Verbindung sie gehören. Da diese Funktion rechenintensiv ist, wird sie nicht beim Erstellen der Datenbank ausgeführt, sondern muss auf eine bestehende Datenbank angewandt werden. Die Berechnung geschieht anhand von *SEQ*- und *ACK*-Nummern, sowie *SYN*-, *ACK*- und *FIN*-Flags.

Ein SYN-Flag signalisiert im TCP-Standard, dass ein System (im folgenden *Client* genannt) eine Verbindung zu einem weiteren System (im folgenden *Server* genannt) herstellen möchte. Dabei schickt er ihm eine zufällig gewählte Sequenznummer mit. Der Server erwidert dies mit einem Paket, bei dem das SYN- und das ACK-Flag gesetzt sind. Hierin enthalten sind die Sequenznummer des Servers, sowie eine ACK-Nummer, die dem Client mitteilt, welche Sequenznummer der Server im nächsten Paket erwartet. Als letzten Schritt des Verbindungsaufbaus sendet der Client ein Paket mit einem gesetzten ACK zurück. Jedes darauf folgende Paket enthält eine SEQ- und eine ACK-Nummer, sowie ein gesetztes ACK-Flag.

[14] Adress Resolution Protocol, siehe [Plu82]

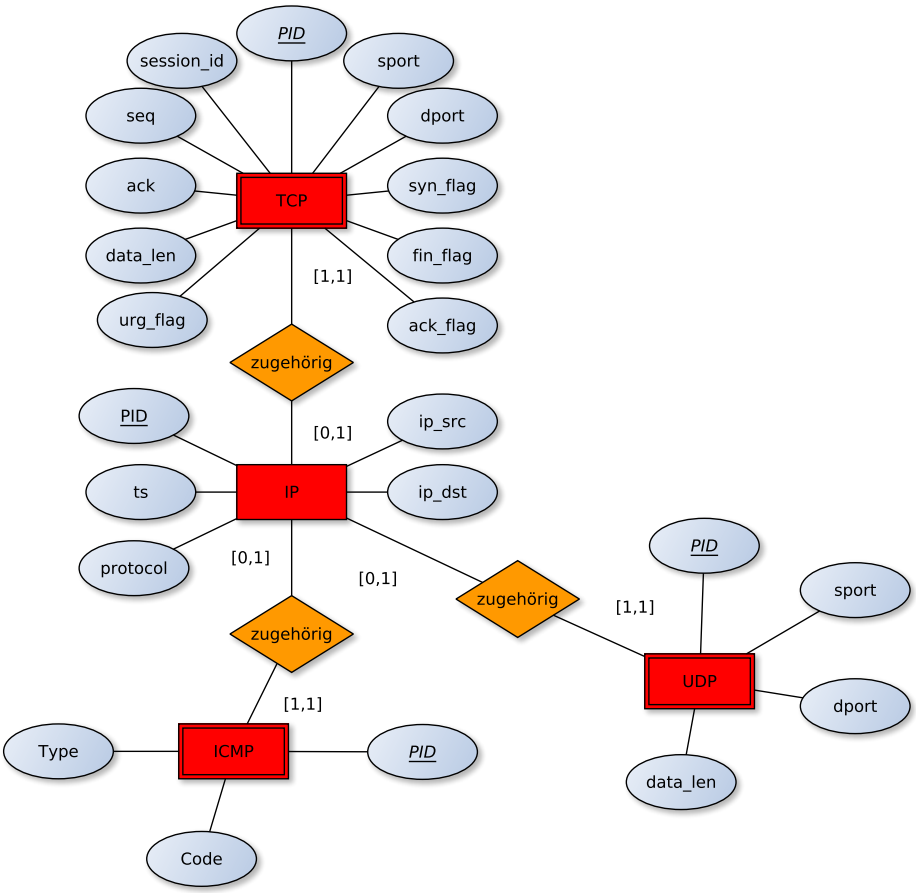

Abbildung 1: ER-Modell der Datenbank

Der Verbindungsabbau funktioniert analog mit FIN- statt eines SYN-Flag. Die Partei, die die Verbindung schließen möchte, sendet ein FIN, welches von einem FIN-ACK erwidert wird und schließlich endet die Verbindung mit einem letzten ACK.

Zum Erkennen der TCP-Verbindungen wird zunächst die temporäre Tabelle *TCP-Session* angelegt, in der jede bereits erfasste Verbindung einen anhand der Session-ID (kurz *SID*) eindeutig identifizierbarenEintrag erhält. Die wichtigsten Spalten dieser Tabelle sind *ip_1*, *ip_2*, *port_1*, *port_2*, sowie *seq_1* und *seq_2*, wobei *1* für die Client-Seite und *2* für die Server-Seite steht. Hierbei ist zu beachten, dass in seq_1 nicht die Sequenznummer des aktuellen Pakets, sondern die nächste erwartete Sequenznummer steht! Diese Tabelle wird grundsätzlich nur für Berechnungen verwendet. Daher wird sie im Anschluss daran gelöscht.

Nun werden alle Einträge in den Tabellen TCP und IP selektiert und es findet ein Verbund über die PID statt. Somit sind alle relevanten Daten zur Berechnung gesammelt und der Algorithmus kann arbeiten. Bei jedem Paket wird überprüft, ob das SYN-Flag gesetzt ist, das ACK-Flag jedoch nicht.

Falls dem so ist, handelt es sich bei dem betrachteten Paket um ein Paket, welches eine Verbindung aufbaut, das heißt es gibt zu dieser Verbindung noch keinen Eintrag in der Tabelle TCP-Session. Dieser Eintrag wird nun angelegt; hierbei ist jedoch wichtig, dass dieser Eintrag in der Spalte *seq_1* nicht die tatsächliche Sequenznummer enthält, sondern dieser Wert um eins erhöht abgespeichert wird. Das liegt daran, dass in *seq_1* immer die nächste erwartete Sequenznummer steht und diese bei einem gesetzten SYN-Flag um eins erhöht wird.

Falls dem nicht so ist, gehört das Paket zu einer bereits bestehenden Verbindung. Daher wird in TCP-Session nach Verbindungen gesucht, in denen die Absender-Adresse des betrachteten Pakets vorkommt und bei denen die Portnummern übereinstimmen. Hier sind grundsätzlich die folgenden Fälle zu betrachten, die hier von der Client-Seite aus geschildert werden. Die Berechnung aus der Server-Perspektive funktioniert analog.

1. Ein Tupel besitzt bei *seq_1* den gleichen Wert, wie die Sequenznummer des aktuellen Pakets. Hier kann das Paket eindeutig dieser Verbindung zugeordnet werden. Daher wird die SID des Pakets auf die SID der Verbindung gesetzt. Zusätzlich wird der Eintrag in TCP-Session geändert, indem die ACK-Nummer des aktuellen Pakets in die Spalte *seq_2* geschrieben wird. Für den Fall, dass bei dem Paket das FIN-Flag gesetzt ist, wird ebenfalls die Spalte *state* auf *1* gesetzt, was einen beginnenden Verbindungsabbau markiert.

2. Ein Tupel besitzt bei *seq_2* den gleichen Wert, wie die ACK-Nummer des aktuellen Pakets. Auch hier kann das Paket eindeutig dieser Verbindung zugeordnet werden. Es findet jedoch nur dann eine Änderung der Tabelle TCP-Session statt, falls wie oben das FIN-Flag gesetzt ist.

3. Keiner der oberen Fälle trifft zu. Somit gehört das aktuelle Paket nicht zu der betrachteten Verbindung und die oben genannten Vergleiche werden bei der nächsten Verbindung durchgeführt.

Sollte das Paket einer Verbindung zugeordnet werden können, werden keine Vergleiche mehr bei anderen Verbindungen durchgeführt, da ein Paket nicht zu mehr als einer Verbindung gehören kann.

Wenn alle TCP-Pakete abgearbeitet wurden, wird die Tabelle TCP_Session gelöscht und die Datenbankverbindung wird geschlossen.

3.2 Plotskripte

Im Rahmen dieser Arbeit wurde eine Reihe von Python-Skripten entwickelt, die mithilfe des Moduls *matplotlib.pyplot*[15] die Ausprägung einiger Eigenschaften der Mitschnitte anschaulich machen. Alle im folgenden Kapitel aufgeführten Graphen wurden durch solche Skripte erstellt.

[15] weitere Informationen unter `http://matplotlib.org/`

4 Evaluation

4.1 Beschreibung der Mitschnitte

Der erste Mitschnitt (welcher ursprünglich im pcapng-Format ausgeliefert worden ist) ist ein anonymisierter Trace, welcher am 17.09.2012 um 08:00:02 Uhr CEST auf dem Webserver der Heinrich-Heine-Universität Düsseldorf angelegt worden ist. Anonymisiert bedeutet hierbei, dass die IP-Adressen zwar verändert worden sind, jedoch eine ursprüngliche IP-Adresse immer auf die gleiche anonymisierte Adresse abgebildet wird. Der Trace umfasst ungefähr 24 Stunden; das letzte Paket wurde am 18.09.2012 um 08:00:28 Uhr CEST erfasst. Der Algorithmus zur Berechnung der TCP-Verbindungen hat 652170 Verbindungen gefunden, die zu 31746 verschiedenen IP-Adressen gehören.

Der zweite Mitschnitt wurde auf einem Server angelegt, auf dem die Webseite der Fachschaft Informatik der Heinrich-Heine-Universität Düsseldorf lag. Das erste Paket ist am 18.07.2012 um 08:17:14 Uhr CEST aufgezeichnet worden, das letzte am 20.07.2012 um 11:22:48 Uhr CEST. Somit umfasst der Trace ungefähr 51 Stunden. Der Algorithmus zur Berechnung der TCP-Verbindungen hat 39358 Verbindungen gefunden, die zu 2159 verschiedenen IP-Adressen gehören.

4.2 Auswertung

4.2.1 Zeitlicher Verlauf und TCP-Portverteilung

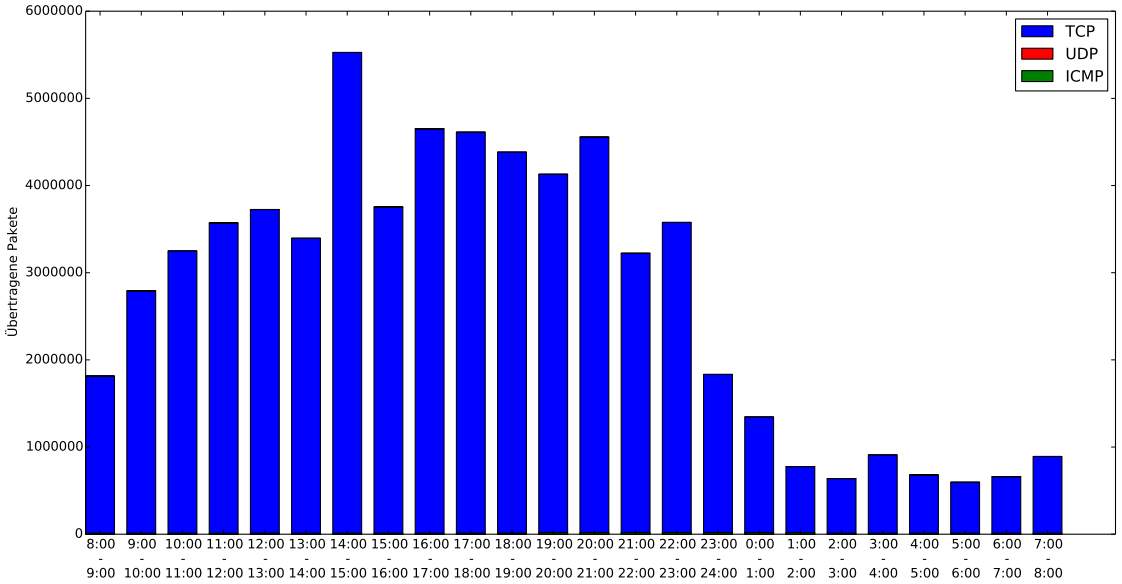

Abbildung 2: Webserver – Zeitlicher Verlauf des Mitschnitts

Abbildung 2 stellt den zeitlichen Verlauf des Webserver-Mitschnitts dar. Deutlich zu erkennen ist hier, dass zwischen 01:00 Uhr und 08:00 Uhr weniger Verkehr stattgefunden hat als zwischen 14:00 Uhr und 15:00 Uhr des Vortages. Insgesamt kann die Zeit zwischen 14:00 Uhr und 21:00 Uhr als Hauptstoßzeit bezeichnet werden, da hier verhältnismäßig viele Pakete unterwegs waren. Auf den ersten Blick fällt ebenfalls auf, dass nahezu der gesamte Verkehr (ungefähr 99,52 %) über TCP stattgefunden hat. Das liegt daran, dass der Mitschnitt auf einem Webserver aufgezeichnet wurde und somit das Anwendungsprotokoll *HTTP*[16] das dominante Protokoll ist (HTTP benutzt TCP als Transportprotokoll).

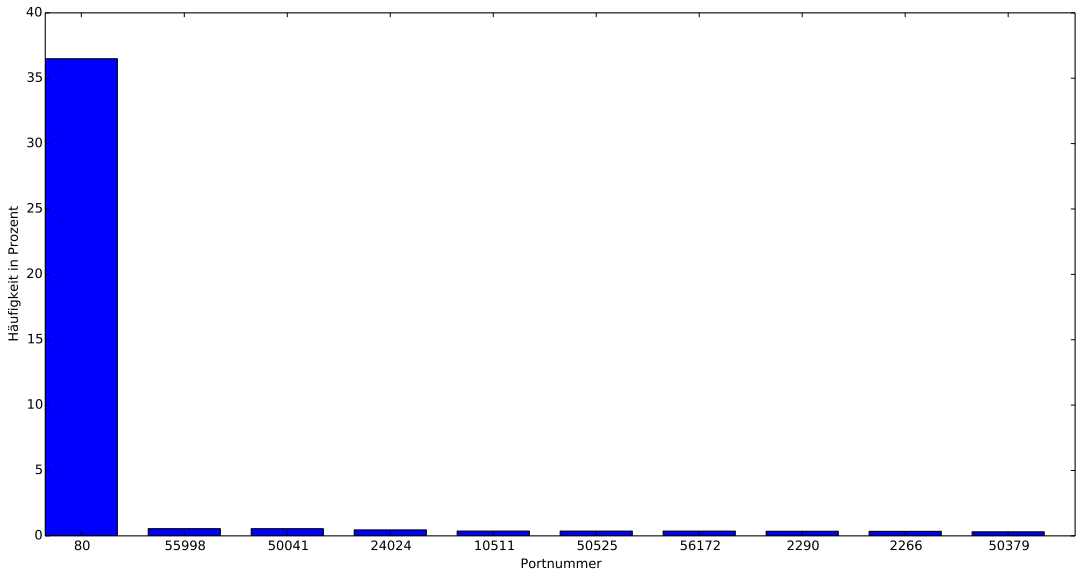

Abbildung 3: Webserver – Verteilung von TCP-Portnummern

Diesen Eindruck bestätigt auch Abbildung 3, in der die zehn häufigsten TCP-Portnummern in diesem Mitschnitt aufgeführt sind. Hier ist deutlich zu sehen, dass die am meisten benutzte Portnummer Port 80 ist. Diese Nummer ist für HTTP reserviert. Bereits die nächsthäufigste Nummer besitzt eine relative Häufigkeit von unter 1%.

[16] Hypertext Transfer Protocol, siehe [FGM+99]

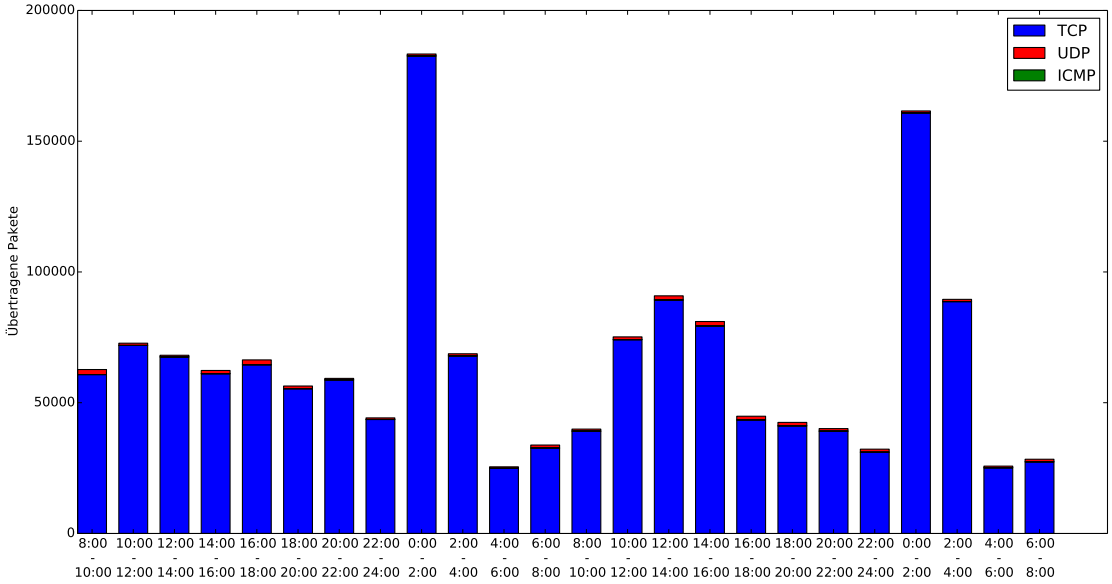

Abbildung 4: Fachschaft – Zeitlicher Verlauf des Mitschnitts

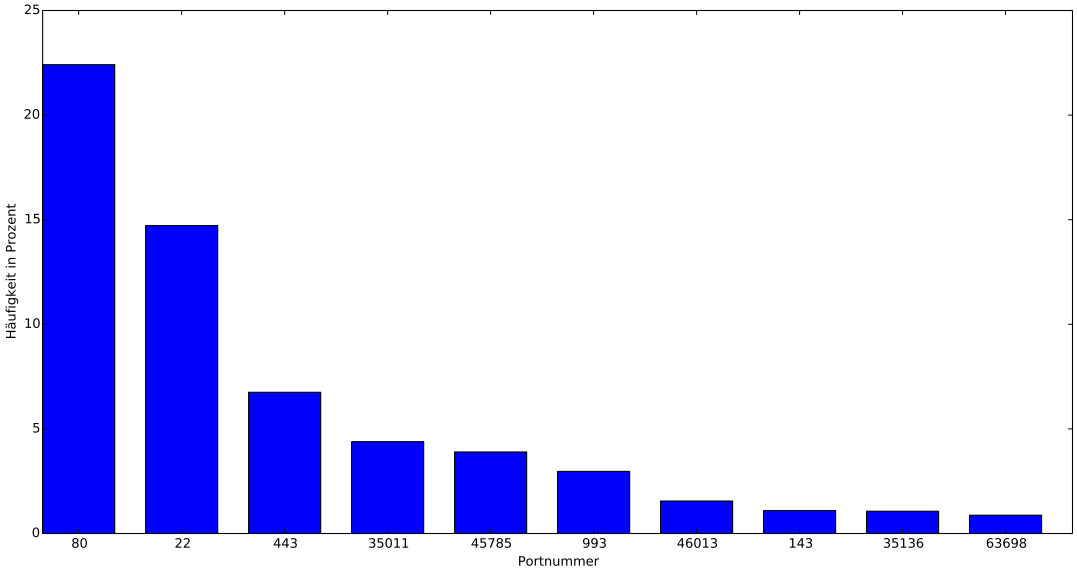

Abbildung 5: Fachschaft – Verteilung von TCP-Portnummern

Abbildung 4 zeigt den zeitlichen Verlauf des zweiten Mitschnitts. Auch hier gibt es deutliche Stoßzeiten zwischen 00.00 Uhr und 02.00 Uhr. Diese sind aber eher ungewöhnlich,

sodass eine tiefergehende Analyse notwendig wurde.

Diese hat ergeben, dass nahezu das gesamte Datenaufkommen zu diesen Uhrzeiten über das Protokoll *SSH* (Port 22)[17] stattgefunden hat. SSH ist ein Anwendungsprotokoll, welches ermöglicht, eine verschlüsselte Verbindung auf ein entferntes System herzustellen. Das spricht dafür, dass um diese Uhrzeiten lediglich Wartungsarbeiten am Server vorgenommen wurden, sodass die eigentlichen Stoßzeiten auf ungefähr 10.00 Uhr bis 20.00 Uhr zu legen sind. Anhand von Abbildung 5 ist ebenfalls erkennbar, dass auf dem Server der Fachschaft Informatik auch andere Protokolle benutzt wurden und dieser kein exklusiver Webserver ist. HTTP ist zwar das dominierende Protokoll; in der Grafik sind aber auch SSH (für Fernwartungen), *HTTPS* (Port 443)[18], *IMAP* (Port 143)[19] und *IMAPS* (Port 993)[20] zu finden.

4.2.2 UDP-Portverteilung

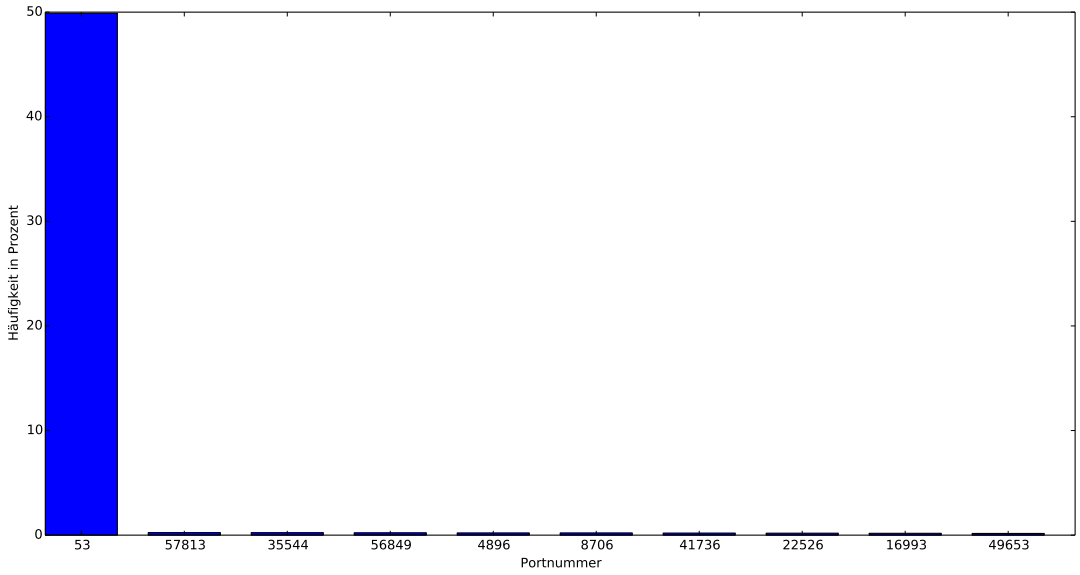

Abbildung 6: Fachschaft – Verteilung von UDP-Portnummern

Die gleiche Verteilung kann man auf UDP-Ebene betrachten. Beim Fachschafts-Server ist die UDP-Verteilung (Abbildung 6) interessanterweise ähnlich einseitig, wie die TCP-Verteilung des Webservers. Deutlich hervorstechend ist *DNS* (Port 53)[21].

DNS ist eines der wichtigsten Protokolle der Anwendungsschicht. Es übersetzt eine für

[17] Secure Shell, siehe [YL06]
[18] HTTP Over TLS, siehe [Res00]
[19] Internet Message Access Protocol, siehe [Cri03]
[20] IMAP Over TLS, siehe [New99]
[21] Domain Name System, weitere Informationen unter [Moc87]

den Menschen lesbare Domain, wie *dbs.cs.uni-duesseldorf.de*, in eine IP-Adresse (in diesem Fall 134.99.112.21), welche von einem Computer interpretiert werden kann. Dies ist nötig, wenn die Adresse nicht im lokalen Cache gespeichert ist. Das System fragt zunächst einen lokalen DNS-Server nach der IP-Adresse des angeforderten Dienstes. Wenn dieser die Adresse ebenfalls nicht auflösen kann, geht der Weg von einem DNS-Root-Server über einen Top-Level-Domain-DNS-Server (in diesem Fall für *.de*) zum autoritativen DNS-Server der Domain, der dann spätestens die IP-Adresse zur Domain kennt.

Dieses Verfahren (*DNS-Lookup*) ist notwendig, wenn ein System eine Verbindung zu einem anderen System aufbauen möchte, den er nicht kennt. Wieso sollte dies jedoch ein Server von sich aus in solch einer Häufigkeit machen?

Dieser Umstand lässt sich wie folgt erklären: Ursprünglich wurde angenommen, dass der Mitschnitt auf dem Server der Fachschaft Informatik selbst angelegt wurde. Weitergehende Analysen haben aber ergeben, dass alle DNS Anfragen zwischen einer (zuerst) unbekannten IP-Adresse und einem DNS-Server, der nicht der hauseigene DNS-Server der Heinrich-Heine-Universität ist, gependelt sind. Nach weiterer Recherche zeigte sich, dass die unbekannte IP-Adresse einer Privat-Person (im Folgenden *Bob* genannt) gehörte. 2012 lag die Webseite der Fachschaft Informatik nämlich noch nicht auf dem Fachschafts-Server (im Folgenden *Majestix* genannt), sondern war getrennt davon auf einer virtuellen Maschine auf dem privaten Server von Bob. Die Fachschaft besaß und besitzt immer noch mehrere Webcams, die sich im Fachschaftsraum befanden und somit an Majestix angeschlossen waren. Die Bilder dieser Webcams waren über die Webseite der Fachschaft erreichbar. Das heißt, dass sich der Server von Bob regelmäßig die Bilder von Majestix heruntergeladen hat, was sowohl den hohen Verkehr zwischen Bob und Majestix, als auch die Häufigkeit der DNS-Lookups erklärt.

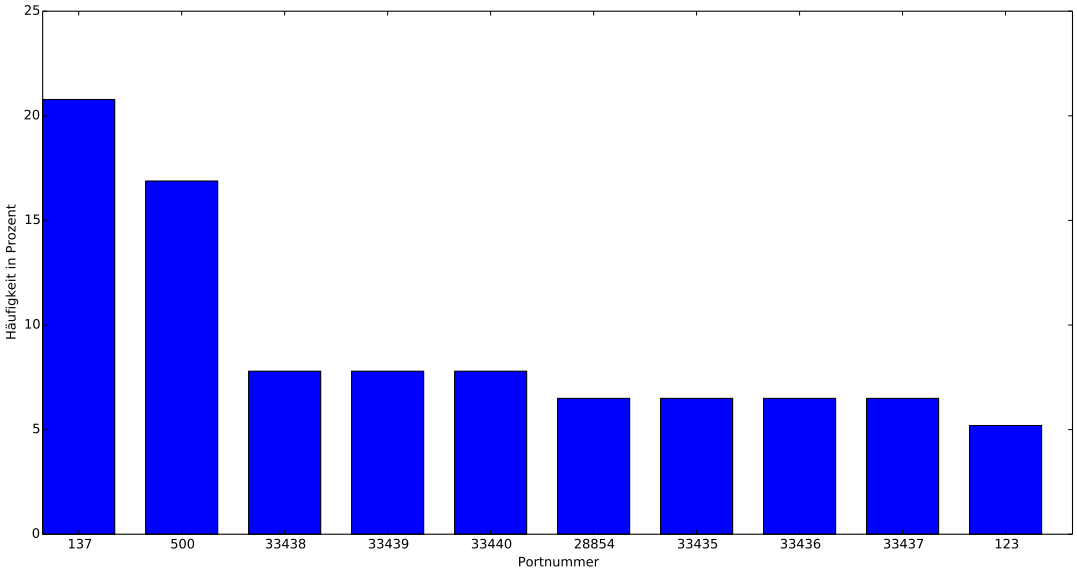

Abbildung 7: Webserver – Verteilung von UDP-Portnummern

Der Webserver-Mitschnitt enthält zwar kaum UDP-Verkehr; der Vollständigkeit halber ist diese Verteilung jedoch in Abbildung 7 trotzdem angegeben. Zu den häufigsten Protokollen gehören hier *NetBIOS Name Service* (Port 137)[22], *Internet Security Association and Key Management Protocol* (Port 500)[23] und *Network Time Protocol* (Port 123)[24].

4.2.3 Anzahl der benutzten TCP-Ports pro IP

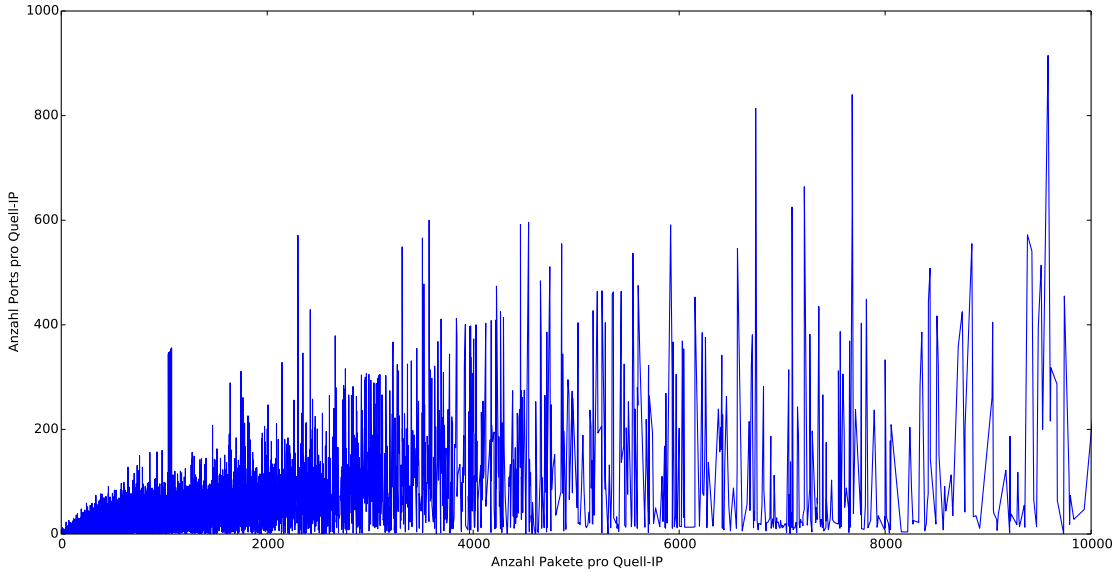

Abbildung 8: Webserver – Anzahl Ports pro Quell-IP

Abbildung 8 und Abbildung 9 stellen die Anzahl der TCP-Ports pro IP-Adresse dar. Bei beiden Graphen ist der gleiche Zusammenhang zu erkennen: Es scheint, als ob mit der Anzahl der Pakete pro IP, auch die Anzahl der benutzten TCP-Ports zunimmt. Dies lässt sich vor allem dadurch erklären, dass bei einem TCP-Verbindungsaufbau eine zufällige Portnummer als Quell-Port ausgesucht wird, während die Zielportnummer bei der gleichen Anwendung grundsätzlich die gleiche ist.

Diese Eigenschaft ist bei beiden Mitschnitten sehr ähnlich ausgeprägt. Wenn man davon absieht, dass der Mitschnitt des Fachschafts-Servers viel weniger Datensätze beinhaltet, weil das Aufkommen des Servers viel kleiner ist, gleichen sich beide Graphen. Diese Eigenschaft des TCP-Protokolls besitzt also eine recht geringe Varianz über mehrere Mitschnitte hinweg.

[22] siehe [Net87]
[23] siehe [MSST98]
[24] siehe [Mil92]

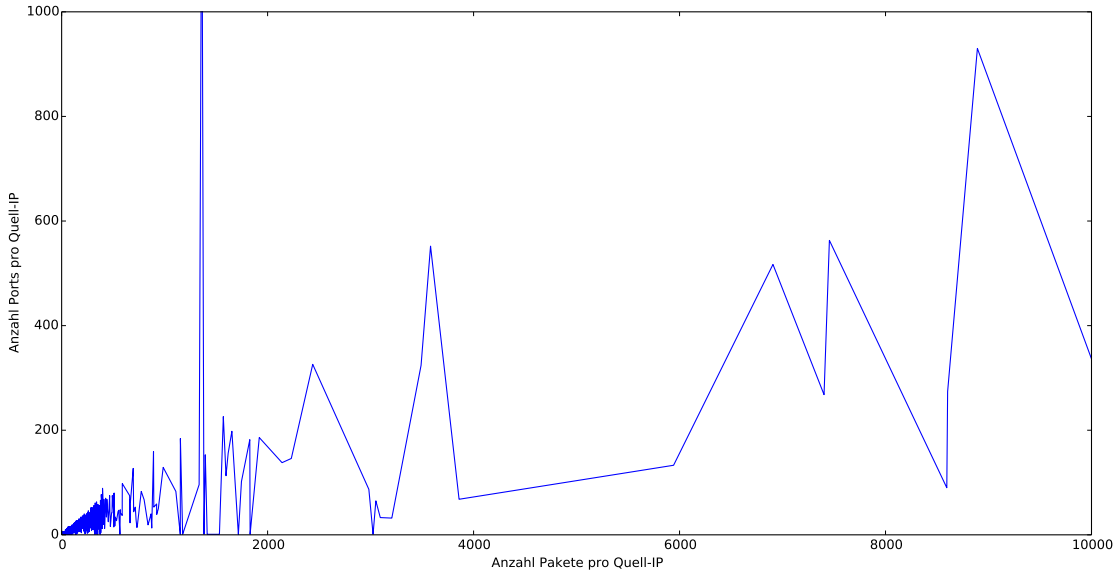

Abbildung 9: Fachschaft – Anzahl Ports pro Quell-IP

4.2.4 Einordnung der geflossenen Datenmengen

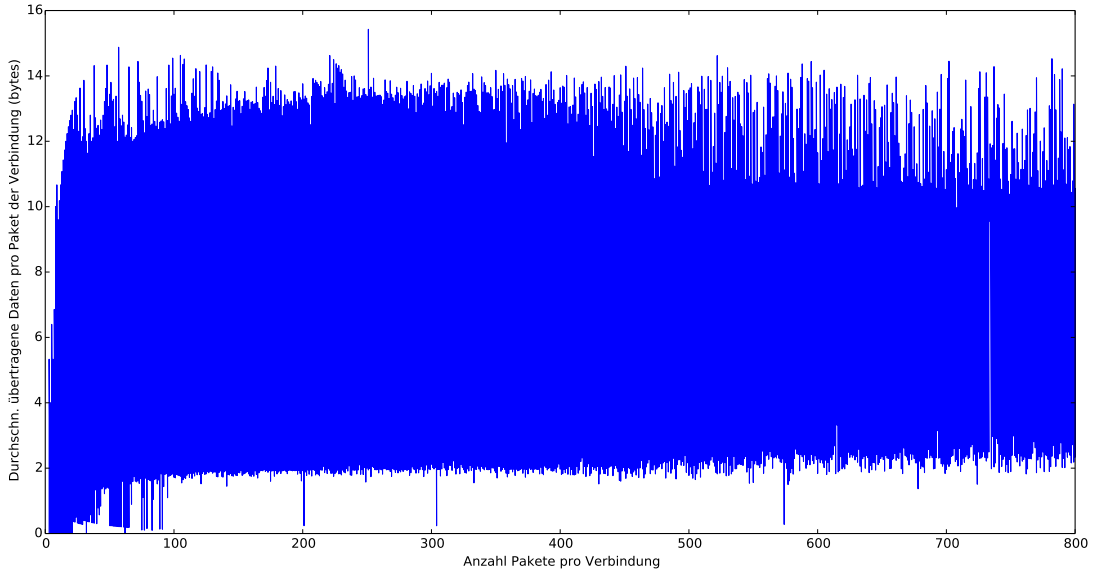

Abbildung 10: Webserver – Durchschnittlich übertragene Daten pro Paket

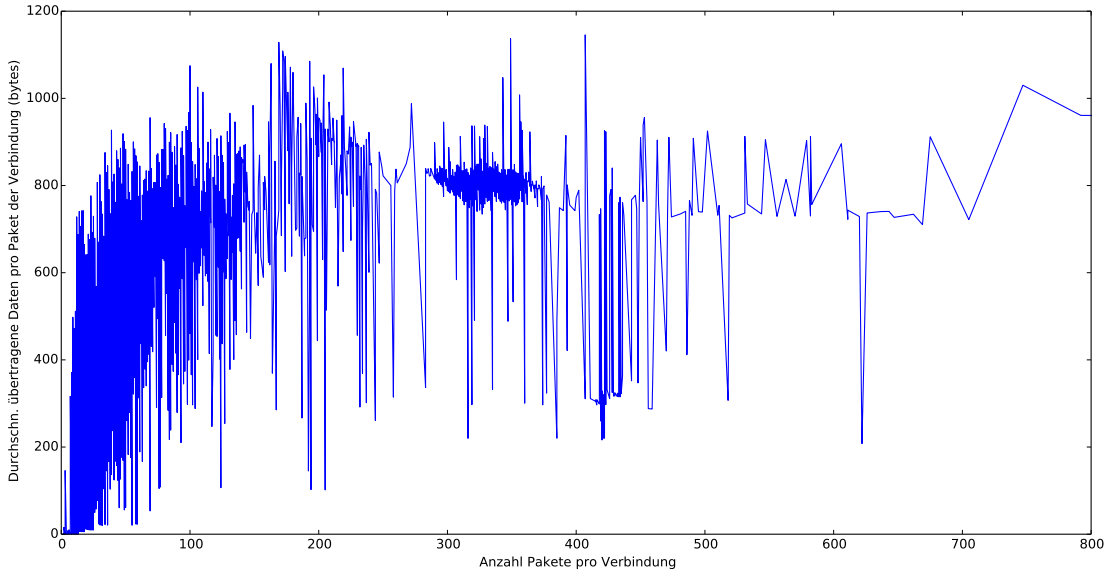

Abbildung 11: Fachschaft – Durchschnittlich übertragene Daten pro Paket

Anhand von Abbildung 10 und Abbildung 11 kann man erkennen, dass die durchschnittlich übertragenen Daten pro Paket im Wesentlichen nicht davon abhängen, wie viele Pakete zu einer Verbindung gehören. Vielmehr hängt die durchschnittliche Größe der Pakete davon ab, welche Anwendungsprotokolle benutzt wurden und wie viele Pakete keine Daten mit sich führten.

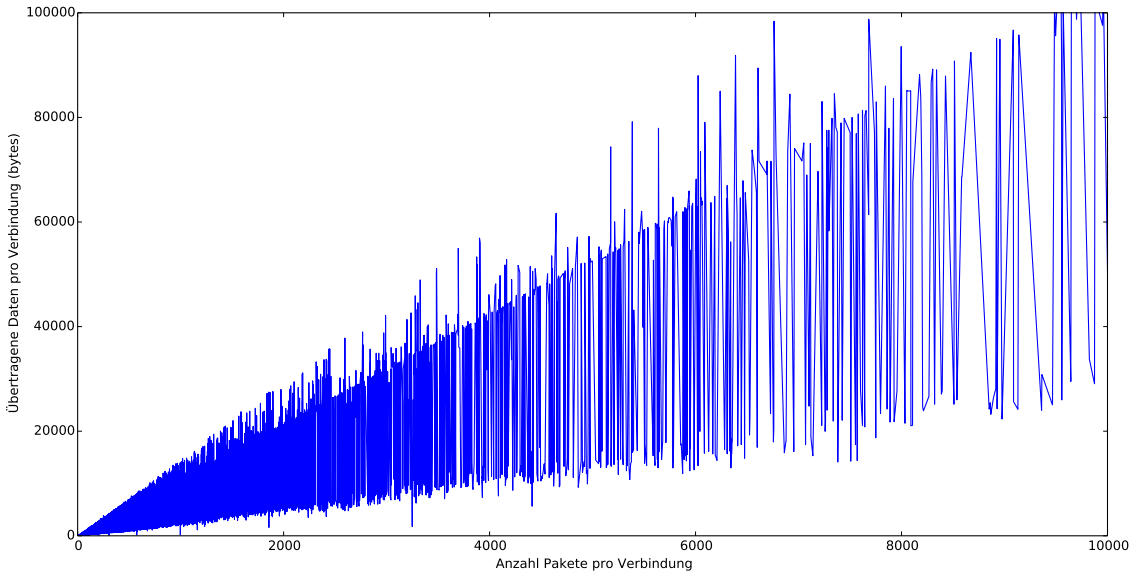

Abbildung 12: Webserver – Übertragene Daten pro Verbindung

Daher nehmen die Daten pro Verbindung proportional zu der Anzahl an Paketen pro Verbindung zu (siehe Abbildung 12 und Abbildung 13). Es ist aber ein entscheidender Unterschied festzustellen: Beim Mitschnitt der Fachschaft sind bereits bei kleineren Verbindungen wesentlich mehr Daten geflossen als beim Mitschnitt des Webservers. Abbildung 10 und Abbildung 11 zeigen, warum dies der Fall ist: Die Pakete des Verkehrs der Fachschaft sind wesentlich „voller“als die Pakete des Webservers; sie enthalten also um ein Vielfaches mehr an Nutzdaten.

Das wiederum könnte daran liegen, dass der Netzwerkverkehr (wie unter 4.2.1 beschrieben) zu einem Großteil aus SSH-Paketen besteht. Diese enthalten oftmals bis zu 1500 Bytes Nutzdaten und sind selten leer, wie das bei HTTP oft der Fall ist, wenn z.B. eine Webseite angefordert wird. Der Client schickt hier vermehrt „ACK-Pakete“, die keine Daten mit sich führen und nur der Bestätigung des Empfangs dienen.

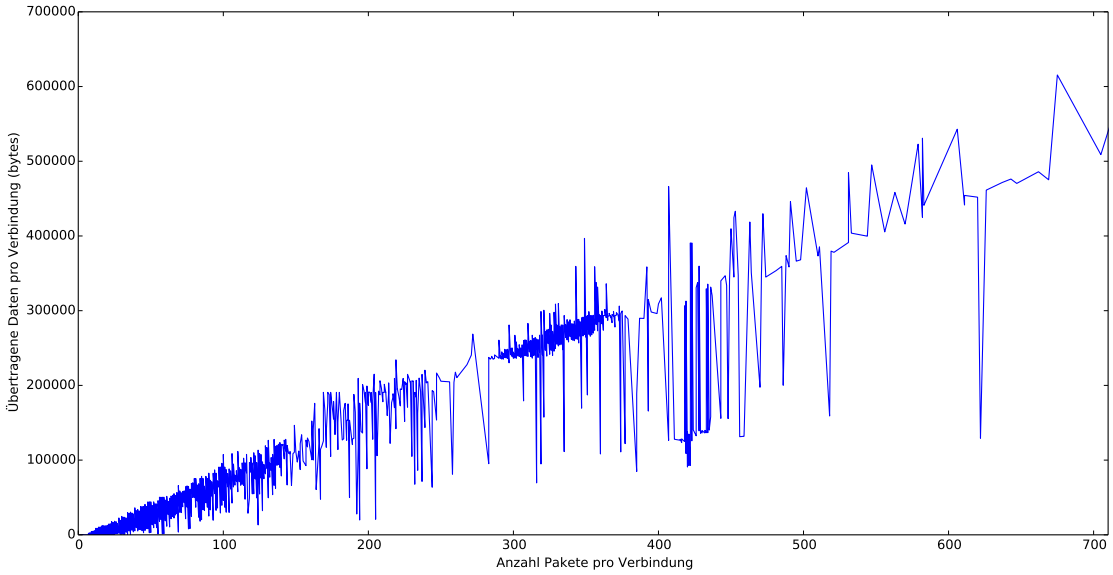

Abbildung 13: Fachschaft – Übertragene Daten pro Verbindung

5 Fazit und Ausblick

In Kapitel 4 wurden zwei Mitschnitte von verschiedenartigen Servern analysiert und die Ergebnisse dieser Analysen verglichen, um allgemeinere Aussagen treffen zu können. Im Folgenden erfolgt die Einordnung der beschriebenen Resultate in den Zusammenhang der Netzwerküberwachung.

Die erste analysierte Eigenschaft war die Stoßzeit eines Servers. Diese besitzt von Mitschnitt zu Mitschnitt eine meist geringe Varianz und lässt sich generell auf zwischen etwa 12.00 Uhr und 20.00 Uhr einordnen. Wenn hier also große Unterschiede auftreten, die nicht durch Wartungsarbeiten etc. erklärbar sind, lohnt sich eine weitergehende Prüfung des Verkehrs.

Ebenso zahlt sich eine Betrachtung der vorkommenden Transportprotokolle, sowie der Anteil an ICMP-Paketen aus. Sollte zum Beispiel bei einem Webserver ein ungewöhnlich hoher Anteil an UDP- oder ICMP-Paketen zu finden sein, könnte das ein Anzeichen auf einen versuchten (*distributed*) *denial-of-service*-Angriff (DoS- bzw. DDoS-Angriff)[25]sein. Bei dieser Form des Angriffs soll erreicht werden, dass ein Netzwerkdienst für seine Nutzer nicht mehr erreichbar ist, indem die gesamten Ressourcen des Dienstes vom Angreifer beansprucht werden. Ein klassischer DoS-Angriff wird hierbei von einem einzelnen System getragen; bei einem DDoS-Angriff sind eine große Zahl von Systemen – wissentlich oder unwissentlich in Form eines Bot-Netzes – beteiligt.

Für die Erkennung eines (D)DoS-Angriffs ist auch eine Analyse der Anzahl der benutzten TCP-Ports pro Quell-IP sinnvoll. Wenn diese Anzahl unverhältnismäßig hoch ist, sodass fast jedes eingehende Paket einer bestimmten IP-Adresse einen neuen zufälligen Port als Quell-Port benutzt, kann zum Beispiel ein SYN-Flood-Angriff stattgefunden haben. Hierbei sendet der Angreifende viele TCP-Pakete zum Verbindungsaufbau; der Dienst beantwortet jedes einzelne mit einem SYN-ACK und wartet auf ein ACK, welches jedoch nicht zurück kommt. Bis das Timeout-Intervall für dieses ACK-Paket abgelaufen ist, muss der Server also Informationen zu jeder eingegangenen Verbindung halten.

Die Beobachtung der durchschnittlich übertragenen Daten pro Paket ist hingegen weniger ertragreich, da diese Eigenschaft eine hohe Varianz aufweist. Von leeren Paketen bis hin zu Paketen, die die maximale Paketgröße (bei Ethernet 1500 Bytes) besitzen, ist hier alles möglich.

Da von der letzten genannten Eigenschaft auch die insgesamt übertragenen Daten pro Verbindung beeinflusst werden, ist auch hier eine nähere Untersuchung im Bezug auf Netzwerküberwachung nicht besonders nützlich.

[25] für weitere Informationen siehe [HR06]

Zukünftige Arbeiten könnten mehr Mitschnitte behandeln und somit die Ergebnisse dieser Arbeit weiter festigen. Außerdem könnten weitere Eigenschaften untersucht werden, um womöglich andere Verfahren zur Netzwerküberwachung zu finden.

Dafür würde sich zum Beispiel eine Auswertung des UDP-Verkehrs eignen, da man anhand der Größe der Antworten möglicherweise *Amplification*-Angriffe erkennen kann. Bei dieser Art von Angriff werden kleine UDP-Pakete mit einer gefälschten Absender-Adresse verschickt, die der Server mit sehr großen Paketen beantworten muss. Darunter zu leiden hat dann die Bandbreite des Systems, dessen IP-Adresse als Absender gewählt wurde.

Zudem könnte näher auf den ICMP-Verkehr eingegangen werden, da eine Tabelle für ICMP-Pakete bereits in der Datenbank existiert. Auf diese wurde in dieser Arbeit jedoch nicht näher eingegangen, da in beiden untersuchten Mitschnitten kaum relevanter ICMP-Verkehr vorhanden war.

Literatur

[Cri03] CRISPIN, M.: *RFC 3501: INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. ftp://ftp.math.utah.edu/pub/rfc/rfc3501.txt. Version: März 2003

[FGM+99] FIELDING, R. ; GETTYS, J. ; MOGUL, J. ; FRYSTYK, H. ; MASINTER, L. ; LEACH, P. ; BERNERS-LEE, T.: *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. ftp://ftp.math.utah.edu/pub/rfc/rfc2616.txt. Version: Juni 1999

[HR06] HANDLEY, M. ; RESCORLA, E.: *RFC 1700: Internet Denial-of-Service Considerations*. ftp://ftp.math.utah.edu/pub/rfc/rfc4732.txt. Version: November 2006

[Mil92] MILLS, David L.: *RFC 1305: Network Time Protocol (Version 3) Specification, Implementation*. ftp://ftp.math.utah.edu/pub/rfc/rfc1305.txt. Version: März 1992

[Moc87] MOCKAPETRIS, P. V.: *RFC 1034: Domain names — concepts and facilities*. ftp://ftp.math.utah.edu/pub/rfc/rfc1034.txt. Version: November 1987

[Mog84] MOGUL, Jeffrey: *RFC 0919: BROADCASTING INTERNET DATAGRAMS*. ftp://ftp.math.utah.edu/pub/rfc/rfc919.txt. Version: Oktober 1984

[MSST98] MAUGHAN, D. ; SCHERTLER, M. ; SCHNEIDER, M. ; TURNER, J.: *RFC 2408: Internet Security Association and Key Management Protocol (ISAKMP)*. ftp://ftp.math.utah.edu/pub/rfc/rfc2408.txt. Version: November 1998

[Net87] NETBIOS WORKING GROUP: *RFC 1002: Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications*. ftp://ftp.math.utah.edu/pub/rfc/rfc1002.txt. Version: März 1987

[New99] NEWMAN, C.: *RFC 2595: Using TLS with IMAP, POP3 and ACAP*. ftp://ftp.math.utah.edu/pub/rfc/rfc2595.txt. Version: Juni 1999

[Plu82] PLUMMER, D. C.: *RFC 826: Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*. ftp://ftp.math.utah.edu/pub/rfc/rfc826.txt. Version: November 1982

[Pos80] POSTEL, J.: *RFC 768: User Datagram Protocol*. ftp://ftp.math.utah.edu/pub/rfc/rfc768.txt. Version: August 1980

[Pos81a] POSTEL, J.: *RFC 791: Internet Protocol*. ftp://ftp.math.utah.edu/pub/rfc/rfc791.txt. Version: September 1981

[Pos81b] POSTEL, J.: *RFC 792: Internet Control Message Protocol*. ftp://ftp.math.utah.edu/pub/rfc/rfc792.txt. Version: September 1981

[Pos81c] POSTEL, J.: *RFC 793: Transmission Control Protocol*. `ftp://ftp.math.utah.edu/pub/rfc/rfc793.txt`. Version: September 1981

[Res00] RESCORLA, E.: *RFC 2818: HTTP Over TLS*. `ftp://ftp.math.utah.edu/pub/rfc/rfc2818.txt`. Version: Mai 2000

[RP94] REYNOLDS, J. ; POSTEL, J.: *RFC 1700: ASSIGNED NUMBERS*. `ftp://ftp.math.utah.edu/pub/rfc/rfc1700.txt`. Version: Oktober 1994

[YL06] YLONEN, T. ; LONVICK, C.: *RFC 4253: The Secure Shell (SSH) Transport Layer Protocol*. `ftp://ftp.math.utah.edu/pub/rfc/rfc4253.txt`. Version: Januar 2006

Abbildungsverzeichnis

Tabellenverzeichnis