UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Bachelorarbeit

Analyse und Klassifikation von Blackhole-Internetverkehr

Patrick Schumacher
Matrikelnummer: 2257624

Abteilung Informatik und Angewandte Kognitionswissenschaft
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

9. September 2013

Betreuer:
Prof. Dr.-Ing. Torben Weis
Dipl.-Inform. Matthäus Wander

Inhaltsverzeichnis

1. Einleitung **1**

1.1. Motivation 1

1.2. Aufgabenstellung 1

1.3. Aufbau der Arbeit 2

2. Grundlagen **3**

2.1. OSI - TCP/IP Modell 3

2.2. Protokolle 4

2.3. PCAP (Packet Capture) 9

2.4. Software-Muster 9

3. Bisherige Ansätze **13**

3.1. Konzept des Blackholes 13

3.2. Charakterisierung des Hintergrundrauschens im Internet 14

3.3. Erneute Untersuchung des Hintergrundrauschens im Internet 15

4. Konzept **17**

4.1. Beschreibung der Beispieldaten 17

4.2. Vorbereitung eines Datensatzes 18

4.3. Ablauf 19

5. Implementierung **25**

5.1. Einlesen von Dateien 25

5.2. Verarbeitung einer Datei 26

5.3. Erfasste Daten einzelner Dateien zwischenspeichern 30

5.4. Datenstrukturen 31

5.5. Zusammensetzen der Zwischenergebnisse 33

6. Analyse und Klassifizierung **35**

6.1. Vorverarbeitung 35

6.2. Analyseergebnisse und Statistiken 36
6.3. Vermittlungsschicht 36
6.4. Transportschicht 38
6.5. Anwendungsschicht 42

7. Fazit **45**

Selbstständigkeitserklärung **47**

A. Anhang **49**
A.1. Analyse und Klassifizierung 49

Literaturverzeichnis **53**

Abbildungsverzeichnis

2.1. ISO/ OSI-Modell, TCP/IP-Modell, Zugehörige Beispiel-Protokolle [TW12] 4
2.2. Der Kopf im IPv4-Protokoll 5
2.3. Der Kopf im UDP-Protokoll 6
2.4. Der Kopf im TCP-Protokoll 7
2.5. Der Kopf im ICMP-Protokoll 8

4.1. Ablauf einer Datei-Verarbeitung 20
4.2. Ablauf eines Erzeugers 22
4.3. Ablauf eines Verbrauchers 23
4.4. Schichtweise-Analyse 24

5.1. Vererbungshierarchie eines als BitTorrent klassifizierten Pakets . . . 29

6.1. Ländercodes bezogen auf die Anzahl Pakete / Quelladresse 36
6.2. Zeitliche Verteilung eingehender Pakete 37
6.3. Arten von Typen und Codes, ICMP 38
6.4. Anzahl der Bytes von Kopf- und Nutzdaten, TCP 39
6.5. Vorkommen der Control-Flags, TCP 40
6.6. Top 10 der Well-Known-Ports, Top 10 der angefragten Ports, TCP . 40
6.7. Zielports und Nutzdatengröße in Byte, UDP 41
6.8. Vermutliche BitTorrent-Anfragen, UDP 43
6.9. Angefragte Methode, HTTP 44

A.1. Top 10 Zielports bei BitTorrent, TCP 49
A.2. Sha1s BitTorrent, TCP 50
A.3. PeerIds BitTorrent, TCP 50
A.4. ReservedExtensionBytes BitTorrent, TCP 50
A.5. Angefragte URLs, HTTP 51
A.6. Anfragender User-Agent, HTTP 51

1. Einleitung

> „Wer eine Schlacht gewinnen will, muß denken, daß er der Sieger ist. Man kann eine Schlacht auch verlieren, wenn man denkt, man ist der Sieger. Aber man kann nie und nimmer gewinnen, wenn man sich für einen Verlierer hält.“ - Roman Polanski

1.1. Motivation

Netzwerkprotokolle und die darauf aufbauende Software bieten wie auch andere Entwicklungen oftmals Angriffsflächen für kriminelle Aktivitäten oder Fehler, welche bei der Veröffentlichung noch nicht absehbar waren. Durch Angriffe oder aber auch durch falsch konfigurierte Systeme und andere unvorhergesehene Ereignisse entstehen große unsinnige Datenmengen. Die Übertragung dieser Daten verbraucht jedoch wertvolle Ressourcen, die es möglichst sparsam und effizient einzusetzen gilt. Deshalb bietet es sich an, anfallenden Netzwerkverkehr zu analysieren und anschließend zu klassifizieren, um mit Hilfe der dabei gewonnen Erkenntnisse bestehende Software zu verbessern oder gegebenenfalls an geeigneten Stellen unerwünschte Daten zu filtern. Die ständige Veränderung des Internets erfordert hierbei jedoch stetig eine erneute Überprüfung und Anpassung der bisherigen Ansätze an aktuelle Gegebenheiten.

1.2. Aufgabenstellung

In dieser Arbeit soll eine Analyse- und Klassifikationssoftware entwickelt werden, welche möglichst modular an neue Gegebenheiten angepasst werden kann. Um dies zu erreichen soll auf bekannte Methoden und Muster der Softwareentwicklung zurückgegriffen werden. Für diese Arbeit steht ein spezieller Datensatz (4.1) von

Internetverkehrsdaten im PCAP-Format zur Verfügung, welcher im Verlauf der letzten Monate am Lehrstuhl der Verteilten Systeme der Universität Duisburg Essen aufgezeichnet wurde. Des Weiteren sollen die während der Entwicklung gewonnenen Erkenntnisse bezüglich des aktuellen Datensatzes mit bisherigen Ergebnissen aus bestehenden Ansätzen in der Literatur verglichen werden. Um eine spätere Analyse mit anderen Datensätzen einfach zu realisieren, liegt ein Schwerpunkt dieser Arbeit darin, ein wiederverwendbares und leicht zu erweiterndes Konzept für die Analyse und Klassifikation von Internetverkehrsdaten aufzuzeigen.

Für die anfallenden Datenmassen gilt es eine effiziente Verarbeitungsstrategie zu entwickeln, sodass diese mit aktuell üblicher Hardware zu bewältigen ist. In einer abschließenden Auswertung sollen die erkannten Muster und Daten aufbereitet und soweit wie möglich interpretiert und analysiert werden.

1.3. Aufbau der Arbeit

Einführend werden in **Kapitel zwei** die für diese Arbeit grundlegend relevanten Protokolle genannt und deren dafür wichtigen Faktoren erörtert. Danach werden einige für die Entwicklung der Software verwendeten Grundlagenkonzepte der Informatik wie das Erzeuger-Verbraucher-Muster beschrieben. In **Kapitel drei** folgt die Beschreibung und Erklärung des Konzeptes „Blackhole“ in Bezug auf die Analyse und Klassifikation von Hintergrundrauschen im Internet, sowie dessen Herkunft und die dazugehörigen Erkenntnisse aus vorausgehenden Arbeiten. **Kapitel vier** beschreibt dann das während der Bearbeitungszeit entstandene Konzept für die Entwicklung einer erweiterbaren Klassifikationssoftware. Darauf folgt in **Kapitel fünf** die Beschreibung einiger Kernaspekte der Implementierung des vorher aufgestellten Konzeptes. Vor dem Ausblick und der Zusammenfassung der in dieser Arbeit gewonnenen Erkenntnisse in **Kapitel sieben**, wird in **Kapitel sechs** der für diese Arbeit zur Verfügung gestellte Beispieldatensatz mit Hilfe der zuvor entwickelten Klassifikationssoftware analysiert.

2. Grundlagen

Um ein besseres Verständnis für gewisse Aspekte dieser Arbeit zu bekommen, werden im Folgenden einige Grundlagen erörtert. Dabei werden große Teile der einzelnen Themen ausgelassen, sofern diese nicht essentiell wichtig für diese Arbeit sind. Bei den später aufgezeigten Protokollen werden dabei die Protokolle der Anwedungsebene ebenfalls ausgeschlossen, da jedes davon immer nur einen speziellen Bereich der Analyse abdeckt und dabei für andere Analysen höchstwahrscheinlich angepasst, ausgetauscht oder durch ein anderes ersetzt werden muss.

2.1. OSI - TCP/IP Modell

Für die Entwicklung der Klassifikationssoftware ist es naheliegend sich an dem in der Praxis oft bewährten TCP/IP-Referenzmodell für Protokolle im Internet zu orientieren. Hierbei durchläuft jedes aufgezeichnete Paket, wie in Abbildung 2.1 zu sehen, die entsprechenden Schichten von unten nach oben. Dadurch wird ein normaler Empfänger simuliert, wobei auf den unteren Schichten allgemeinere Meta-Daten von den Paketen gesammelt werden, während die höheren Ebenen präzisere Inhaltsanalysen vornehmen.

Zum Vergleich ist hierbei das genormte ISO/OSI-Modell neben dem TCP/IP-Modell dargestellt, in welchem jedoch einige Aufgaben auf andere Schichten als im TCP/IP-Modell verteilt sind. Dies resultiert aus der Tatsache, dass viele Aufgaben, die die Schichten fünf und sechs im ISO/OSI-Modell übernehmen, in der Praxis oftmals gar nicht vorkommen. In manchen Fällen werden diese jedoch, entgegen dem Gedanken der gewünschten Kapselung, auch in der nächsthöheren Schicht behandelt. Das Auslassen dieser Schichten im TCP/IP-Modell bedingt dabei jedoch die Verschiebung der Behandlung in die höheren Schichten innerhalb der Klassifikationssoftware.

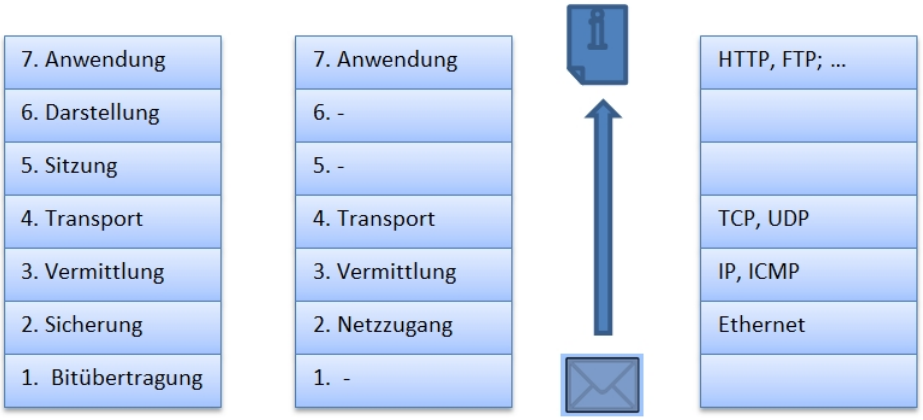

Abbildung 2.1.: ISO/ OSI-Modell, TCP/IP-Modell, Zugehörige Beispiel-Protokolle [TW12]

2.2. Protokolle

Im folgenden Abschnitt folgt die exemplarische Erklärung einiger grundlegend relevanter Protokolle, die in dieser Arbeit primär untersucht werden. Dabei werden IPv4 und die vorkommenden Nutzdatenprotokolle ICMP, UDP und TCP mit deren für diese Arbeit relevanten Details beschrieben und auf die entsprechenden Standards verwiesen.

2.2.1. IPv4 (Internet Protocol Version 4)

Aufgrund der immer noch sehr großen Verbreitung von IPv4 [Pos81b] im Vergleich zu IPv6 ist es das für diese Arbeit überwiegend relevante Protokoll. Von der Google Inc. existiert eine regelmäßig aktualisierte Statistik [Goo13], nach der die weltweite Verwendung von IPv6 zum aktuellen Zeitpunkt noch bei unter zwei Prozent liegt. IPv6 wird auch deshalb nicht näher betrachtet, da es nur in marginaler Menge getunnelt in den Beispieldaten identifiziert werden konnte (siehe Analyse (6)).

Im *Versionsfeld* des in Abbildung 2.2 gezeigten IPv4-Kopfes steht daher die Versionsnummer vier. Danach folgt die *Länge des Kopfteiles* des Pakets, wobei dieser mindestens 20 und maximal 60 Byte umfassen kann. Die ersten 20 Byte bilden den immer existierenden Header eines jeden Pakets, während die restlichen 40 Byte optional sind. Das Feld *Dienstklasse* bietet dem Sender die Möglichkeit, eine gewünschte Priorisierung für das Paket genauer festzulegen. Danach folgt die

Paketlänge, welche sich über den Header und die Nutzdaten erstreckt. Die Felder *Identifikation*, *DF (Nicht Fragmentieren)*, *MF (Mehr Fragmente)* sowie der *Fragment-Versatz (Offset)* bieten im IPv4 Protokoll die Möglichkeit, zu große Pakete aufzuteilen (siehe auch 2.2.1). Das nächste Feld im IP-Kopf gibt die *Lebenszeit* eines Paketes an, welches dazu dient, endlos umlaufenden Paketen vorzubeugen. Daraufhin folgt das *Protokollfeld*, welches den Protokoll-Typ der Nutzdaten genauer spezifiziert. Beispiele sind hier TCP (2.2.3) oder UDP (2.2.2). Abschließend verbleibt noch die *Prüfsumme*, welche zur Integritätssicherung verwendet wird.

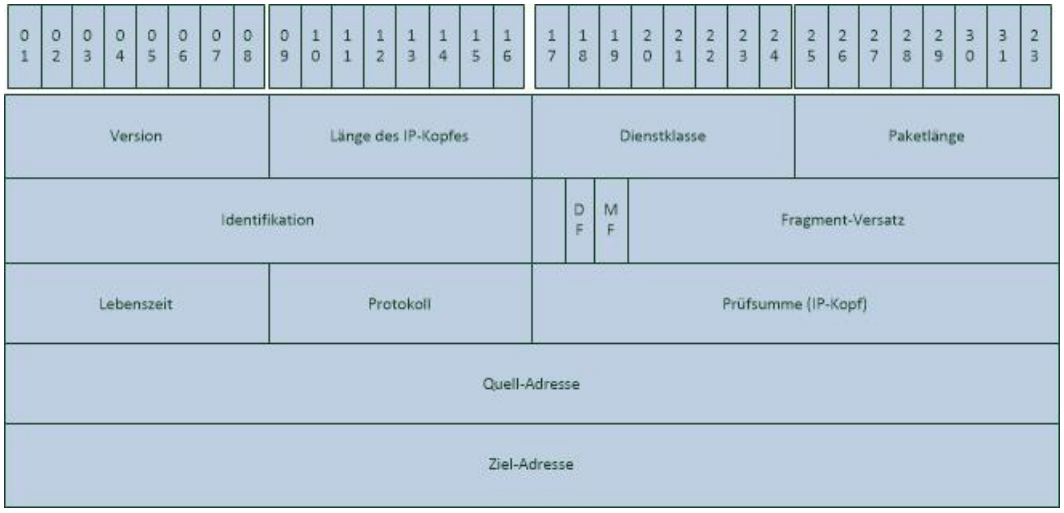

Abbildung 2.2.: Der Kopf im IPv4-Protokoll

Fragmentierung

Das *DF-Bit* gibt an, ob ein Paket fragmentiert werden darf. Sobald das *DF-Bit* auf eins gesetzt ist, darf das Paket nicht fragmentiert werden. Sofern es auf Null gesetzt ist, kann der Router das Paket für den weiteren Weg in mehrere kleinere Pakete aufteilen. Im Falle einer Fragmentierung erhalten alle Teilfragmente dieselbe Identifikationsnummer und bei jedem außer dem letzten ist das *MF-Bit* auf eins gesetzt. Durch eine Null beim letzten Fragment weiß der Empfänger, dass keine weiteren Fragmente mehr folgen. Da das IPv4 Protokoll nicht sicherstellt, ob Fragmente in der richtigen Reihenfolge ankommen, benötigt der Empfänger für das Zusammensetzen der Pakete abschließend noch den *Fragment-Versatz*. Dieser zeigt die Position im ursprünglichen Paket, an der das aktuelle Fragment beginnt.

2.2.2. UDP (User Datagram Protocol)

Das User Datagram Protocol [Pos80] ist im Vergleich zu anderen Protokollen ein schlankes und damit effizienteres Netzwerkprotokoll. Diese Vorteile gehen dabei jedoch auf Kosten der Funktionalität, da es, anders als zum Beispiel das weiter unten beschriebene Transmission Control Protocol, keinerlei Garantien über die Zustellung von Paketen macht. Auch andere Funktionen wie der Schutz vor Überlastung einer Verbindung oder die Priorisierung von Paketen sind nicht durch UDP selbst zu erreichen. Ein UDP-Paket besteht dabei aus einem Kopf- und einem Nutzdatenteil. Im Kopfteil, dem sogenannten Header, befinden sich der *Quell-* und *Ziel-Port*, die *Paket-Länge* und eine optionale *Prüfsumme*.

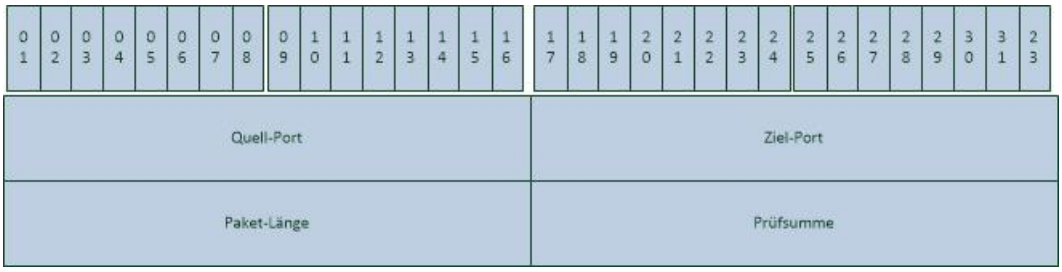

Abbildung 2.3.: Der Kopf im UDP-Protokoll

Da es seitens des Protokolls keinerlei Sitzungsfunktionalität gibt, entfallen hier leere Steuerungs- und Kontrollnachrichten. UDP-Pakete können bei der Aufzeichnung der Daten daher einfach passiv aufgezeichnet werden.

2.2.3. TCP (Transmission Control Protocol)

Das Transmission Control Protocol [Pos81c] ist ein verbindungsorientiertes, paketvermittelndes Transportprotokoll für Netzwerkpakete, welches unter anderem die Zustellung von Paketen sicherstellt. Dabei wird es verbindungsorientiert genannt, da es zwischen Sender und Empfänger einen eindeutig identifizierbaren Kanal öffnet, welcher durch die vier Merkmale *Quell-Adresse:Quell-Port* und *Ziel-Adresse:Ziel-Port* eindeutig bestimmt ist. Allerdings ist dieser Kanal nur sinnbildlich gemeint, da es immer noch ein paketvermittelndes Protokoll ist, bei dem die Pakete zeitversetzt und auf unterschiedlichen Wegen zu ihrem Ziel gelangen können.

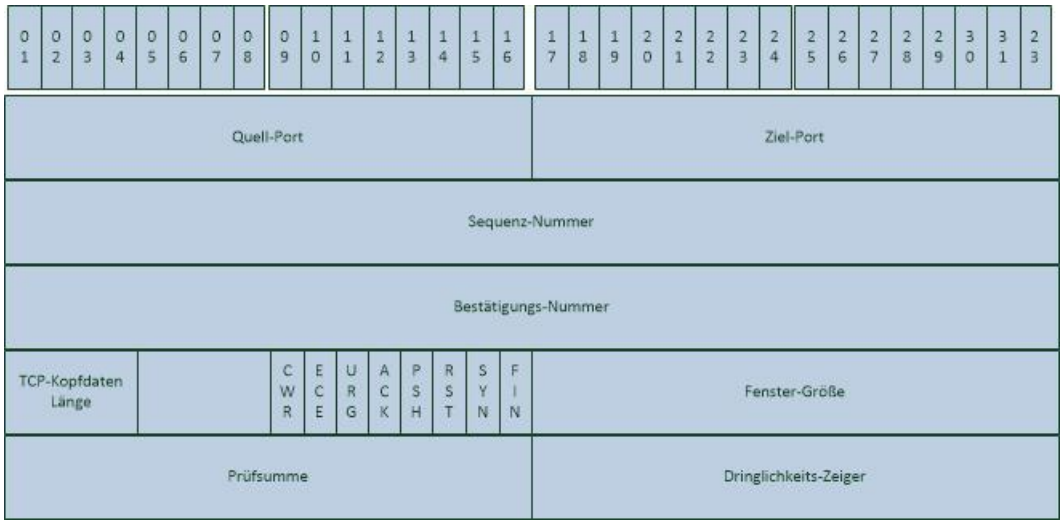

Abbildung 2.4.: Der Kopf im TCP-Protokoll

Der Header eines jeden TCP-Paketes besteht mindestens aus den in Abbildung 2.4 dargestellten Feldern und kann durch einen Optionsteil erweitert werden. Auf den Kopfteil folgen optional die Nutzdaten des Paketes für die nächsthöhere Schicht. Ein Paket ohne Nutzdaten dient unter anderem der Steuerung und Kontrolle der Verbingungsherstellung und Trennung.

Die *Sequenz-* und *Bestätigungs-Nummer* eines TCP-Paketes werden in Verbindung mit den acht *Control-Flags* für den Auf- und Abbau einer TCP-Sitzung (2.2.3) verwendet. Des Weiteren dienen die Nummern der Sicherstellung der korrekten Zustellung von Paketen. Fehlende Fragmente können so identifiziert und die Gegenstelle gegebenenfalls zum erneuten Senden veranlasst werden.

Das Feld *TCP-Kopfdaten-Länge* spezifiziert die Länge des TCP-Headers inklusive der Länge des variablen Optionsteils.

Der *Dringlichkeitszeiger* und die *Fenster-Größe* stellen die Flusskontrolle im TCP-Protokoll dar, welche in dieser Arbeit aber keine Betrachtung findet. Mit dieser kann ein überlasteter Empfänger der Gegenstelle signalisieren, dass er die Menge an Daten nicht verarbeiten kann, woraufhin diese ihre Sendegeschwindigkeit entsprechend anpassen sollte, damit keine Daten verloren gehen.

TCP-Sitzung

Eine TCP-Sitzung wird durch den so genannten „Drei-Wege-Handshake“ initialisiert. Hierbei sendet der initiierende Partner eine leere TCP-Nachricht, bei der im Kopfteil das *SYN-Flag* und eine *Sequenznummer* gesetzt sind. Die Gegenstelle beantwortet diese Nachricht bei Akzeptanz der Verbindung jeweils mit einer Nachricht in der das *ACK-Flag* und die *Bestätigungs-Nummer* gesetzt sind. Letztere ist hierbei in der Regel die um eins erhöhte *Sequenz-Nummer* aus dem Eröffnungsgesuch. Der ursprüngliche Sender antwortet darauf ebenfalls mit einer Nachricht bei der das *ACK-Flag* gesetzt und die *Sequenz-* sowie die *Bestätigungs-Nummer* inkrementiert wurden. Hierbei können gleichzeitig bereits Nutzdaten übermittelt werden, wodurch der „Drei-Wege-Handshake“ abgeschlossen und die Verbindung für weitere Übertragungen genutzt werden kann. Um das Auftreten eines Fehlers oder das Ablehnen einer Verbindung anzuzeigen, kann in den Control-Flags das *RST-Bit* gesetzt werden. Eine jedoch bestehende Verbindung kann von beiden Seiten jederzeit mittels eines gesetzten *FIN-Bits* geschlossen werden. Auf ein gesetztes *FIN-Bit* wird ein dreischrittiger Verbindungsabbau, vergleichbar mit dem Aufbau, eingeleitet. Eine Verbindung kann dabei jedoch auch nur einseitig beendet werden, sodass eine Verbindung weiterhin mit nur einer Sende- und Empfangsrichtung möglich ist.

2.2.4. ICMP (Internet Message Control Protocol)

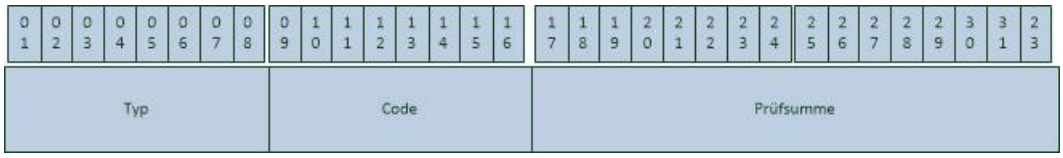

Abbildung 2.5.: Der Kopf im ICMP-Protokoll

Das Internet Message Control Protocol [Pos81a] sollte von jedem Router, sofern er das IP-Protokoll verwendet, unterstützt werden. Es dient zum Austausch von Status- und Kontrollnachrichten, mit deren Hilfe eine bessere Netzqualität erreicht werden soll.

Der ICMP-Kopf besteht dazu, wie in Abbildung 2.5 zu sehen, aus den drei Feldern *Typ*, *Code* und einer *Prüfsumme*. Der *ICMP-Typ* kann zum Beispiel „Ziel nicht erreichbar“ oder „Zeit überschritten“ lauten, während der darauffolgende Code den

Fehler genauer spezifiziert. Mit dem Typ „Ziel nicht erreichbar“ kann zum Beispiel der Code „Port nicht erreichbar“ kombiniert werden. In den auf den Header folgenden Daten befindet sich oftmals eine Kopie der ursprünglichen IP-Nachricht. Durch diese Status- und Fehlermeldung kann der Sender gegenebenenfalls versuchen seine Pakete über eine andere ihm bekannte Route zu versenden oder dem ursprünglichen Absender einen nicht behebbaren Fehler signalisieren. Der jeweilige enthaltene Fehlercode und Typ eines Paketes kann dabei jedoch auch, je nach Ausgestaltung, Hinweise auf die Absicht des ursprünglichen Senders offen legen.

2.3. PCAP (Packet Capture)

PCAP ist wie viele andere Protokolle in dieser Arbeit ein in der Praxis häufig eingesetztes Format zum Speichern von aufgezeichneten Netzwerkdaten. Libcap und Wincap sind dabei die entsprechend unter Linux beziehungsweise Windows verwendeten Programmbibliotheken, welche die Verwaltung der betroffenen Netzwerkhardware übernehmen und ihre aufgezeichneten Daten als *.pcap-Dateien speichern. PCAP selbst hat dabei im Sinne der Standards der Internet Engineering Task Force (IETF) den aktuellen Status „Draft-Standard“, was in etwa mit dem Begriff Entwurf übersetzt werden kann. Dabei kann ein Entwurf vollständig fertig spezifiert sein, für das Erreichen einer höheren Stufe innerhalb dieser Klassifikation bedarf es weiterer Überprüfungen. Ein Draft-Standard ist zum Überprüfen bzw. zum Besprechen durch die Internetcommunity freigegeben und sofern niemand Einspruch erhebt, wird dieser zum vollen Standard erhoben. [AB88, vgl S.2]

2.4. Software-Muster

Diese Sektion beschreibt die verwendeten Software-Muster, welche die Modularität, Wartbarkeit und einfache Erweiterbarkeit der Klassifikationssoftware ermöglichen. Dabei wird auch eine Methodik zur Synchronisation der später im Konzept aufgeführten Prozesse aufgezeigt.

2.4.1. Model-View-Controller (MVC)

Das MVC-Muster dient der Trennung von Anwendungsdaten, Darstellung dieser Daten und Verarbeitung der Nutzereingabe. Es versucht die drei Schichten unabhängig voneinander zu machen, sodass jede dieser Schichten ohne Modifikation der anderen Schichten austauschbar ist. Dies dient einer einfacheren Wartbarkeit genauso wie der Kapselung von Daten und Funktionen in unterschiedliche Komponenten. Dabei bedient es sich dem Beobachter-Muster, wobei das datenumfassende Modell von den datenanzeigenden Präsentationsschichten auf Veränderung beobachtet wird.

2.4.2. Singleton

Das Singleton-Muster versucht nur einmalig verfügbare Schnittstellen im gesamten Programm verfügbar zu machen. Dabei wird zum Beispiel der Zugang zu einer Datenbank einmalig instanziiert und diese Instanz ist dann programmweit einheitlich zu erreichen. Dabei können so einfach Zugriffskontrollen durchgeführt werden und die Entscheidung, welche Art Unterklasse letztendlich für den Zugriff verwendet werden soll, kann zur Laufzeit festgelegt werden. Dadurch lassen sich zum Beispiel verschiedene Datenbankframeworks innerhalb eines Programms über eine Schnittstelle ansprechen.

2.4.3. Erzeuger-Verbraucher

Das Erzeuger-Verbraucher-Muster dient in der Softwareentwicklung dazu, einen begrenzten Speicher beim Einsatz von mehreren Prozessen möglichst effizient auszunutzen. Dabei versucht es die beteiligten Prozesse so zu synchronisieren, dass es bei Zugriff auf eine gemeinsame Datenstruktur weder zu einem Über- noch zu einem Leerlauf des Speichers kommt, wodurch die Gefahr eines stockenden Systems verringert werden kann.

Eine klassische Situation, bei der von der einen Seite erzeugte Objekte in einer gemeinsamen Datenstruktur abgelegt und diese auf der anderen Seite wieder entnommen und verbraucht werden, kann durch den Einsatz dieses Musters überwacht und kontrolliert werden, damit keine der beteiligten Parteien die Überhand

gewinnt. Sollte eine der beiden Seiten zu schnell arbeiten, wird diese solange abgeschaltet oder gedrosselt, bis wieder Platz für neue Objekte ist oder ob äquivalent wieder Daten zum Entnehmen vorhanden sind.

3. Bisherige Ansätze

In diesem Kapitel werden bisherige Ansätze aus der Literatur aufgezählt und beschrieben. Dem vorwegnehmend werden die für diese Arbeit relevanten Teile des Blackholings spezifiziert. Die in den folgenden Abschnitten vorgestellte Technik des Filterns und indirekt auch die des aktiven Antworten auf eingehende Nachrichten wird im Kapitel Konzept (4) aufgegriffen und an die aktuellen Gegebenheiten angepasst.

3.1. Konzept des Blackholes

Um eine Charakterisierung des Hintergrundrauschens im Internet durchzuführen, bietet es sich an, den eingehenden Netzwerkverkehr hinter eigentlich nicht vergebenen IP-Adressen zu analysieren. Unter diesen Adressen ist eigentlich kein normaler Netzwerkverkehr zu erwarten, sodass dort speziell solche Angriffe und Daten eingehen, die automatisiert versuchen große Adressbereiche mit Schadprogrammen zu infizieren oder nach bekannten Schwachstellen suchen, die auf nicht regelmäßig aktualisierten Systemen leicht ausgenutzt werden können.

Der ursprüngliche Gedanke des Blackholes ist dabei, die eingehenden Daten von Angreifern automatisiert zu analysieren um so Erkentnisse über deren Vorgehensweise und über aktiv ausgenutzte Schwachstellen zu gewinnen. Dabei kommen in einem solchen Adressbereich keine normalen Daten vor, sodass normale Fehlverbindungsversuche aufgrund technischer Probleme oder auch andere produktive Daten nicht mit den zu klassifizierenden vermischt werden [PYB+04, vgl S.1].

3.2. Charakterisierung des Hintergrundrauschens im Internet

Der erste bekannte Versuch, das Hintergrundrauschen im Internet zu klassifizieren, geht auf den Artikel 'Characteristics of Internet Background Radiation' [PYB+04] aus dem Jahre 2004 zurück.

Zunächst wurde dort evaluiert, dass auch auf eigentlich nicht vergebenen IP-Adressen ständig große Datenmengen anfallen, auf die eigentlich kein Empfänger wartet. Die dort anfallenden Daten stammen häufig von Angriffsversuchen, welche zum Beispiel automatisiert ganze Adressbereiche nach Systemen mit offenen Schwachstellen untersuchen. Des Weiteren sind dort auch Antwortpakete zu finden, die aus Angriffen mit gefälschten Sendeadressen resultieren, sogenannte „Backscatter“. Neben der generellen Konzeption des Blackholings umfasst ein Kernaspekt des Artikels das Kategorisieren und Filtern der anfallenden Daten, um vor allem neuartige Angriffe von bereits bekannten unterscheiden zu können. Hierzu wurden jedoch mehrere Millionen nicht vergebene IP-Adressen überwacht, was das strikte Filtern alleine aufgrund der schieren Datenmenge erforderlich machte. (vgl. [PYB+04, S.1])

3.2.1. Filtern der Daten

Das Filtern der Daten hat die Aufgabe, die anfallenden Datenmassen zu verringern und gleichzeitig die Vielfalt der Daten zu erhalten, um dennoch alle Arten von eingehendem Verkehr zu erfassen. Als Messgröße der Datenmasse wurden hierbei 30.000 eingehende Pakete pro Sekunde genannt, aus welchen es relevante Daten zu extrahieren galt. Um dies zu realisieren, wurde dabei die Methodik der Quell-Ziel basierten Filterung angewendet. Unter der Annahme, dass es sich bei den eingehenden Daten um automatisierte Angriffs- und Verbindungsverfahren handelt, kann man davon ausgehen, dass eingehende Daten von einer Quelle bei allen Empfängern gleich oder stark ähnlich vorkommen. Daher reicht die einmalige Analyse der Daten von einer Quelladresse in den meisten Fällen aus, sodass Anfragen von der selben Quelle an anderen eingehenden Adressen nicht wieder betrachtet werden müssen. (vgl. [PYB+04, S.2f])

3.2.2. Aktives Antworten

Vor dem zweiten Kernaspekt, dem aktiven Antworten auf eingehende Anfragen, wurde bei der passiven Überwachung der eingehenden Daten festgestellt, dass es sich bei vielen eingehenden Verbindungen um TCP-SYN-Pakete handelte. Um jedoch von TCP-Verbindungen überhaupt analysierbare Nutzdaten zu erhalten, musste auf eingehende Anfragen zunächst einmal aktiv geantwortet und eine gültige TCP-Verbindung hergestellt werden. Die daraus resultierende Erkenntnis der Notwendigkeit von Antworten auf eingehende Verbindungen wurde anschließend auch auf höhere Protokollschichten ausgeweitet, sodass aktive Antwortframeworks für zum Beispiel HTTP, NetBIOS und andere damals verbreitete Protokolle entwickelt werden sollten. Auf das Thema des aktiven Antwortens wird dabei im Kapitel Analyse (6) und ebenfalls im Fazit (7) detailierter eingegangen.

3.2.3. Vergleich und Klassifikation

Mit Hilfe der eben beschriebenen Techniken, aber auch durch einfache statistische Auswertungen der Daten konnte innerhalb des Artikels gezeigt werden, dass die Menge an schädlichem Traffic nicht marginal ist und dieser weitere Beachtung geschenkt werden sollte. Dabei konnten eingehende Daten teilweise bekannten Würmern und Verfahren zugeordnet werden, während andere Daten noch nicht ausreichend behandelt werden konnten. Es wurde ebenfalls aufgezeigt, dass sich das Hintergrundrauschen im Internet teilweise täglich oder zumindest proportional zu neuen weit verbreiteten Technologien im Internet verändert und eine ständige Überwachung erforderlich ist.

3.3. Erneute Untersuchung des Hintergrundrauschens im Internet

Auf die vorherige Arbeit (3.2) aufbauend, wurde in dem Artikel 'Internet Background Radiation Revisited' [WKB+10] eine Wiederholung der Klassifizierung und Analyse durchgeführt. Durch die strukturelle Veränderung des Internets und den Einsatz neuer Software, war rund sechs Jahre später eine erneute Überprüfung der bisherigen Konzepte und Ergebnisse erforderlich.

3.3.1. Ähnlichkeiten und Änderungen

Zunächst einmal stimmt das Fazit dieses Artikels, wobei die Masse des Hintergrundrauschens im Internet einen allgegenwärtigen, vielfältigen und sich ständig verändernden Charakter aufweist, mit dem aus dem vorherigen Artikel überein. Dabei überholt dessen Wachstum jedoch das des produktiven Datenverkehrs im Internet. Im Vergleich zu der ersten Arbeit wurde außerdem eine Erhöhung des SYN- und ein Rückgang des SYN-ACK-Traffics festgestellt. In den verschiedenen Adressblocks, die untersucht wurden, gab es jedoch signifikante Unterschiede, die laut der Autoren jedoch auf strukturellen und lokalitätsbezogenen Gründen basieren und nicht von algorithmischer Natur herrühren. [WKB+10, vgl. S.12]

4. Konzept

Dieses Kapitel umfasst die konzeptuelle Entwicklung der Klassifikationssoftware. Grundlage sind dabei einerseits die in der Literatur (3) verwendeten Techniken und Erkenntnisse, andererseits legen der vorgegebene Datensatz (4.1) und die in der Einleitung beschriebene Aufgabenstellung (1.2) gewisse Eckpunkte bereits von vornherein fest. Es wird ein kompletter Durchlauf der Analyse inklusive notwendiger Vor- und Nachbereitungsschritte analysiert. Das Konzept legt damit die Vorgaben für die Entwicklung der Software fest, welche im Kapitel Implementierung (5) beschrieben wird.

4.1. Beschreibung der Beispieldaten

Für diese Arbeit wurde der Netzwerkverkehr auf 90 nicht vergebenen IP-Adressen über einen Zeitraum von mehreren Monaten aufgezeichnet. Dazu wurde der ein- und ausgehende Datenverkehr mitgeschnitten und im PCAP-Format (2.3) abgelegt. Dabei wurden ausgehende Verbindungen mit Ausnahme von TCP-ACK-Antworten an der Firewall des Lehrstuhls für Verteilte Systeme blockiert. Die technische Konzeption des TCP-Protokolls (2.2.3) bedingte diese Vorgehensweise, da nur so die Nutzdaten einer TCP-Verbindung erfassbar waren. Dies entspricht teilweise dem Konzept des aktiven Antwortens, wobei in dieser Arbeit keine Antworten auf der Anwendungsebene gesendet wurden.

Die für diese Arbeit zu Grunde liegenden Daten waren dabei in sieben in der Größe unterschiedlichen PCAP-Dateien hinterlegt, welche insgesamt rund 153 GB umfassten.

4.2. Vorbereitung eines Datensatzes

Ein Analysedurchlauf der Klassifikationssoftware ist in mehrere logisch voneinander getrennte Abschnitte unterteilt. Dabei sind die im Folgenden beschriebenen Vorbereitungsschritte teilweise in die Software integriert, sodass zum Beispiel eine Bedienung durch andere Anwender oder die Verarbeitung weiterer Datensätze möglich ist. Für die aktuelle Analyse reichte die Vorbereitung mit Hilfe der im Kapitel Implementierung (5) beschriebenen Hilfsmittel jedoch aus.

4.2.1. Filtern der Daten

Da im Verlauf der Aufzeichnung der Netzwerkdaten (4.1) teilweise auch ausgehende Pakete mit aufgezeichnet wurden, müssen diese herausgefiltert werden. Die ausgehenden Nachrichten wurden dabei an der Firewall des Lehrstuhlnetzes nur dann nach außen hin durchgelassen, insofern es sich um TCP-ACK-Antworten bezogen auf ein vorher eingehendes TCP-SYN-Paket gehandelt hat. Die also nach außen hin gar nicht existenten Pakete gilt es bei der Verarbeitung der Daten herauszufiltern. Auch die ausgehenden TCP-ACK-Antworten lassen sich dabei jedoch entfernen, da diese immer ein identisches Schema aufweisen und somit als Annahme für jede eingehende Anfrage angesehen werden können. Hierbei lässt sich leicht erkennen, dass für weitere Analysen andere Filter notwendig sind. Eine zeitlich eingeschränkte Betrachtung oder das Herausfiltern von reinen Steuerungs- und Kontrollnachrichten im TCP-Protokoll, um so die Datenmenge zu verkleinern, ist dafür ein einfaches Beispiel. Denn wie auch in der Analyse des TCP-Protokolles (6.4.1) gezeigt werden konnte, bestehen die Datensätze zu großen Teilen aus TCP-Anfragen ohne Nutzdaten.

4.2.2. Aufsplitten großer Dateien

Um die Masse an Daten überhaupt verarbeiten und dabei wertvolle Informationen extrahieren zu können, müssen große PCAP-Dateien in mehrere kleinere Dateien aufgeteilt werden, da ansonsten das Sammeln der Analyseergebnisse die Speichergrenzen des Auswertungssystems überschreiten würde. Um dennoch eine zusammenhängende Analyse großer Datenmengen zu ermöglichen, sollte die Klassifikationssoftware mehrere Dateien in einen logischen Zusammenhang bringen können.

Dazu müssen die Ergebnisse aus den einzeln analysierten Dateien vor einer Interpretation wieder zusammengefügt werden. Prinzipiell ist dadurch eine Untersuchung beliebiger Datenmengen möglich. Hierbei ist die oberste Grenze jedoch immer die maximal mögliche Größe der zusammengesetzten Ergebnisse, wobei man auch aus den einzelnenen Teilanalysen bereits Erkenntnisse über die verarbeiteten Daten gewinnen kann.

Die Aufteilung großer Dateien in kleinere Teile ermöglicht des Weiteren eine bessere Auslastung des Auswertungssystems, da bei paralleler Verarbeitung mehrerer Dateien ein langsamer Datenspeicher oder eine aufwändige Analyse ausgeglichen werden kann. Dabei hängt die Größe der Aufteilung nicht nur von dem zur Verfügung stehenden Auswertungssystem ab. Eine Analyse von zum Beispiel sitzungsbasierten Protokollen kann nur dann nennenswerte Erkenntnisse offenlegen, sofern die relevanten Teile einer Sitzung innerhalb eines Bereiches der Daten liegen, die die Klassifikationssoftware zusammenhängend verarbeiten kann.

4.3. Ablauf

Nach der Vorbereitung, bei welcher die Dateien zumindest in eine für das System verarbeitbare Größe gebracht werden müssen, folgt in diesem Abschnitt die Betrachtung der Analyse einer solchen verkleinerten Datei. Diese umfasst dabei die Speicherung von Meta-Daten, die Verarbeitung mit Hilfe des Erzeuger-Verbraucher-Musters sowie die Zwischenspeicherung der Analyseergebnisse einer einzelnen Datei. Abschließend folgt die Betrachtung der notwendigen Zusammensetzung der Zwischenschritte zu einem verwertbaren Ergebnis.

4.3.1. Meta-Daten von PCAP-Dateien

Von jeder eingelesenen Datei werden Meta-Daten erfasst, welche zusammen mit den Analyseergebnissen der einzelnen Schichten für die Auswertung im Kapitel Analyse (6) verwendet werden. Dazu werden von jeder Datei der *Name* und die *Anzahl der enthaltenen Pakete* erfasst. Der Name dient der eindeutigen Identifizierung, anhand welcher auch die temporären Analyseergebnisse identifiziert und zwischengespeichert werden.

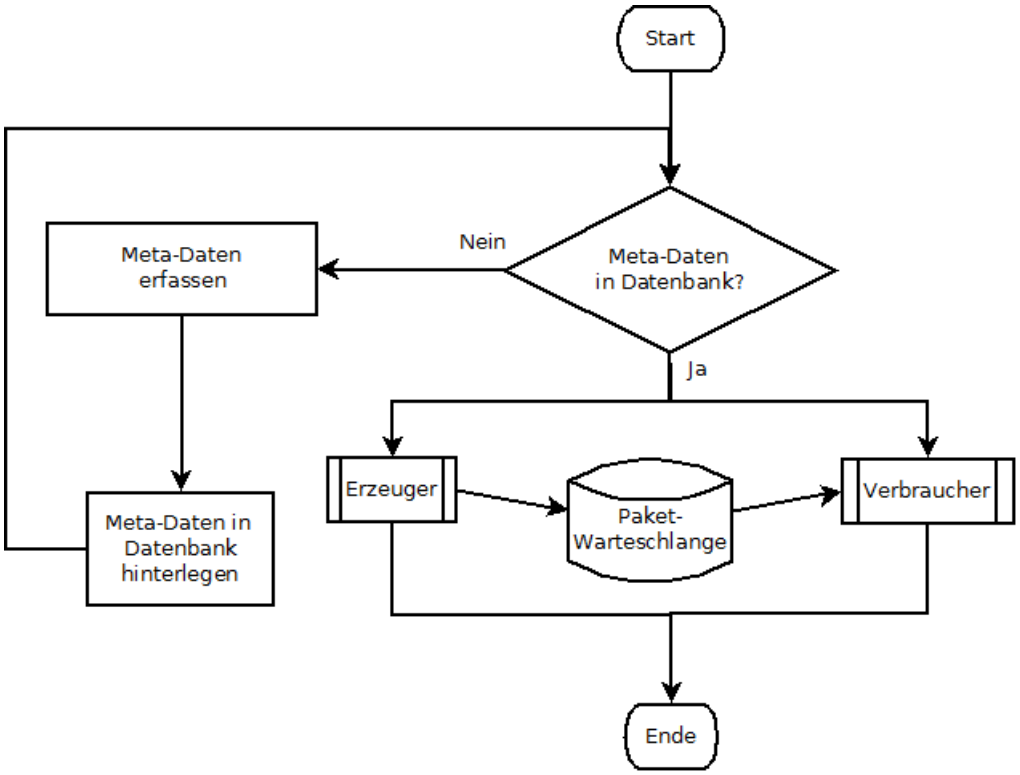

Abbildung 4.1.: Ablauf einer Datei-Verarbeitung

4.3.2. Verarbeitung einer Datei

Abbildung 4.1 zeigt das Erfassen der Meta-Daten und die Aufteilung einer Datei in das Erzeuger-Verbraucher-Muster. Je nach Auslastung des Datenspeichers und der CPU kann so die Systemleistung einer der konkurrierenden Parteien innerhalb der Speichergrenzen zugewiesen werden. Sollte das Einlesen der Datei die Speichergrenze des Systems erreichen, wird der erzeugende Thread solange blockiert, bis der Verbraucher genügend Pakete verarbeitet hat. Umgekehrt kann die volle Systemleistung auch dem erzeugenden Thread überlassen werden, sollte das Verarbeiten die Schlange komplett entleert haben.

4.3.3. Erfasste Daten einzelner Dateien zwischenspeichern

Sobald ein Datensatz vollständig eingelesen und verarbeitet ist, müssen die Zwischenergebnisse aus dieser Datei auf einem Datenspeicher temporär persistiert werden, da die Klassifikationssoftware ansonsten nur kleine Datenmengen in einen

logischen Zusammenhang bringen könnte. Von jeder klassifizierten Datei wird anschließend ein gleichnamiger Ordner in einem temporären Verzeichnis erstellt, in welchem die Ergebnisse der einzelnen Analyseschichten gespeichert werden. Bei der Implementierung einer neuen Verarbeitungsschicht werden dabei die Dateinamen als eindeutige Identifizierung verwendet. Hierbei müssen Schichtübergreifende eindeutige Namen vergeben werden, da diese zur Aggregation der dateiübergreifenden Ergebnisse verwendet werden.

4.3.4. Erzeuger und Verbraucher

Der Erzeuger wird, wie im Ablaufplan in Abbildung 4.2 zu sehen, eine Datei iterativ einlesen und versuchen, von jedem gültigen Ethernetrahmen ein Paket zu generieren. Ein Paket besteht dabei aus den *Nutzdaten*, der eindeutigen *Paket-Identifikation* und einem *Zeitstempel*, welcher den Eingang des Paketes anzeigt. Bei der Paketgenerierung werden dabei gleichzeitig vordefinierte Filter angewendet, damit zum Beispiel die ausgehenden Pakete (4.1) die Warteschlange gar nicht erst blockieren. Sollte der zur Verfügung stehende Speicher trotzdem zur Neige gehen, wartet der Erzeuger mit dem weiteren Einlesen und überlässt dem Verbraucher den Vortritt. Sobald die Datei vollständig abgearbeitet ist, muss dies dem entsprechenden Verbraucher mitgeteilt werden, damit dieser nach dem letztmaligen Entleeren der Schlange nicht erneut auf Pakete wartet. Der Verbraucher ist deshalb in zwei verschiedene Modi unterteilt, welche im Ablaufplan in Abbildung 4.3 sequentiell durchlaufen werden. Der erste ist solange aktiv, bis der Erzeuger signalisiert, dass er keine weiteren Pakete mehr in die Schlange legen wird. Dabei blockiert der Verbraucher sobald sich keine weiteren Paketen in der Warteschlange befinden und überlässt dem Erzeuger die vorhandene Systemleistung. Der zweite Modi dient der restlichen Entnahme der nach dem Beenden des Erzeugers verbleibenden Pakete in der Schlange. Jedes aus der Schlange entnommene Paket wird dabei der mehrschichtigen Analyse zugeführt. Jede Schicht definiert dabei selbst, wie die extrahierten Informationen nach dem vollständigen Durchlauf der aktuellen Datei zu speichern sind.

Nachdem alle Pakete dieser Datei verarbeitet wurden, löst der Verbraucher die entsprechende Speicherung der Zwischenergebnisse der einzelnen Schichten aus. Dies wird erst am Ende einer jeden Datei durchgeführt, da so innerhalb jeder Schicht auch Informationen über mehrere oder alle Pakete dieses Teildatensatzes

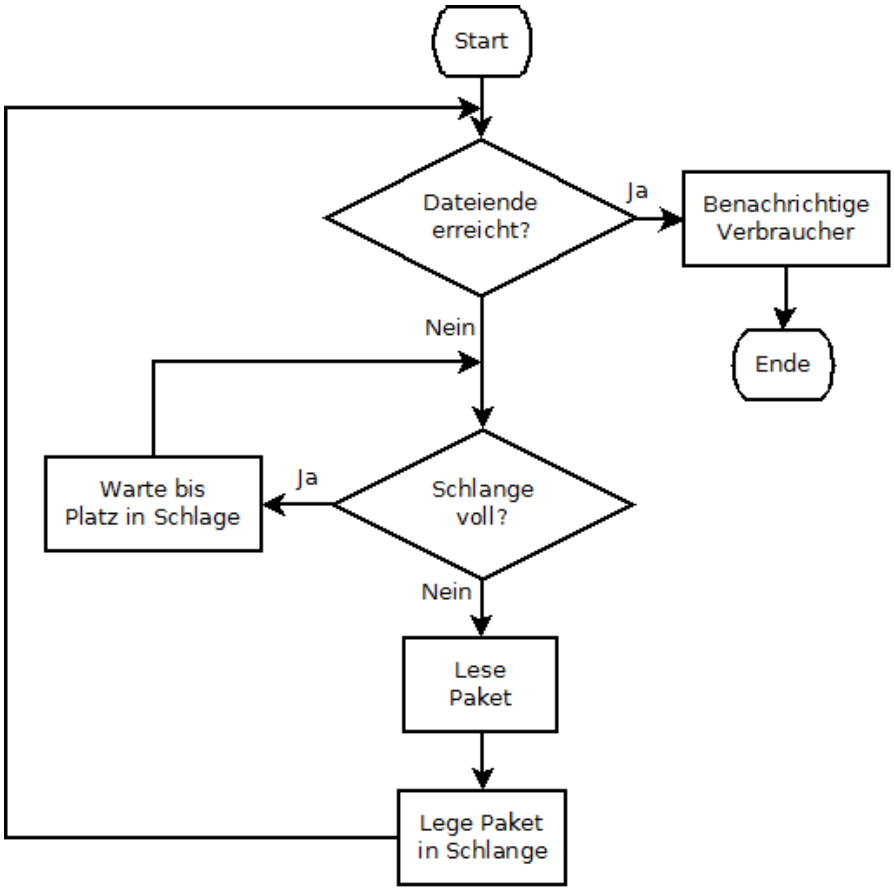

Abbildung 4.2.: Ablauf eines Erzeugers

hinweg verglichen werden können. So können innerhalb eines Verbrauchers auf IP-Ebene zum Beispiel alle Quell-Adressen gesammelt werden.

4.3.5. Schichtweise Interpretation eines Paketes

Sobald ein Verbraucher ein Paket aus der Warteschlange entnommen hat wird, wie in Abbildung 4.4 zu sehen, versucht das Protokoll der Vermittlungsschicht zu identifizieren. An dieser Stelle ist hierbei nur die Unterscheidung zwischen IPv4 und IPv6 zu treffen, wobei IPv6 noch kaum eine Relevanz hat, da es nur in getunnelter Form innerhalb der Daten vorkam (siehe Analyse, 6.3). Eine Erweiterung ist an dieser Stelle in Anlehnung an die Muster für IPv4 ergänzbar, die grundlegenden Funktionen können hierbei wieder verwendet werden.

In Abbildung 4.4 ist dabei zu sehen, dass nach dem Erkennen eines IPv4-Rahmens versucht wird die nächste Schicht, also zum Beispiel TCP, UDP oder ICMP, zu er-

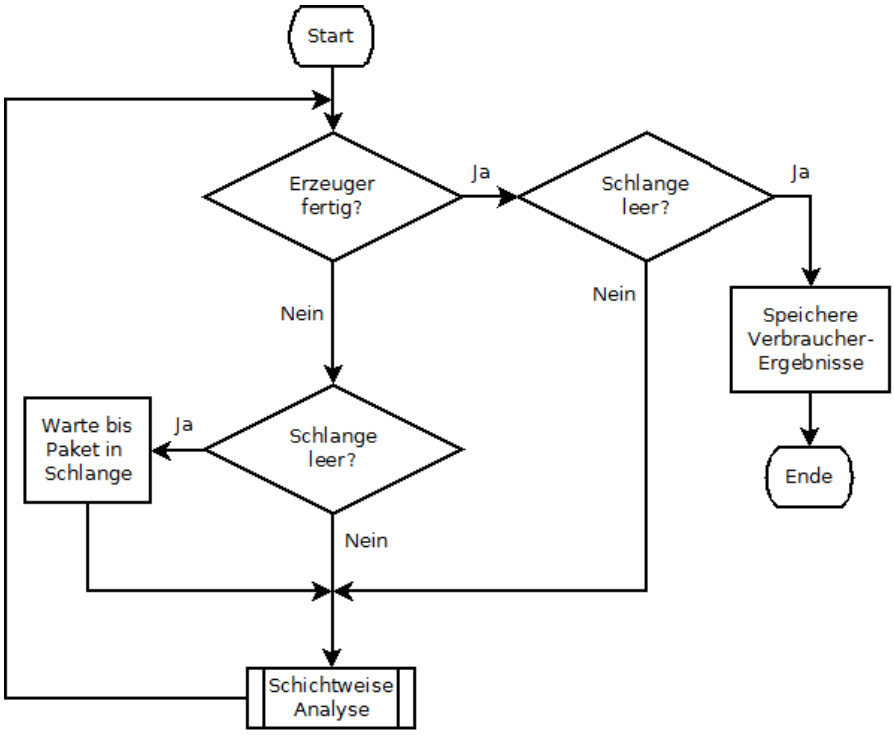

Abbildung 4.3.: Ablauf eines Verbrauchers

mitteln. In jeder dieser höheren Schichten wird hierbei wiederum nach gekapselten Protokollen aus der Anwendungsschicht wie HTTP oder DNS gesucht. Letztendlich werden auf der höchsten Schicht genauere Analysen, wie die Extraktion der angefragten URLs einer DNS-Anfrage oder des Host-Feldes einer HTTP-Anfrage, durchgeführt.

Auf allen durchlaufenen Schichten wird dabei versucht, neben dem reinen Zählen auch andere Meta-Daten wie die Nutzdatengröße oder ähnliches zu erfassen. Hierzu werden unter anderem auch die im Kapitel 5.4 erwähnten Datenstrukturen verwendet.

4.3.6. Zusammensetzen der Zwischenergebnisse

Für eine Auswertung der Daten müssen die temporär abgelegten Dateien abschließend aggregiert werden. Jede zwischenspeichernde Schicht muss dabei neben dem Erkennen, Behandeln und Zwischenspeichern der Daten also auch das spätere Zusammenfügen der Daten definieren. Dabei gilt es Überschneidungen innerhalb der Dateien sinnvoll aufzulösen, sodass am Ende nur eindeutige Ergebnisse in den

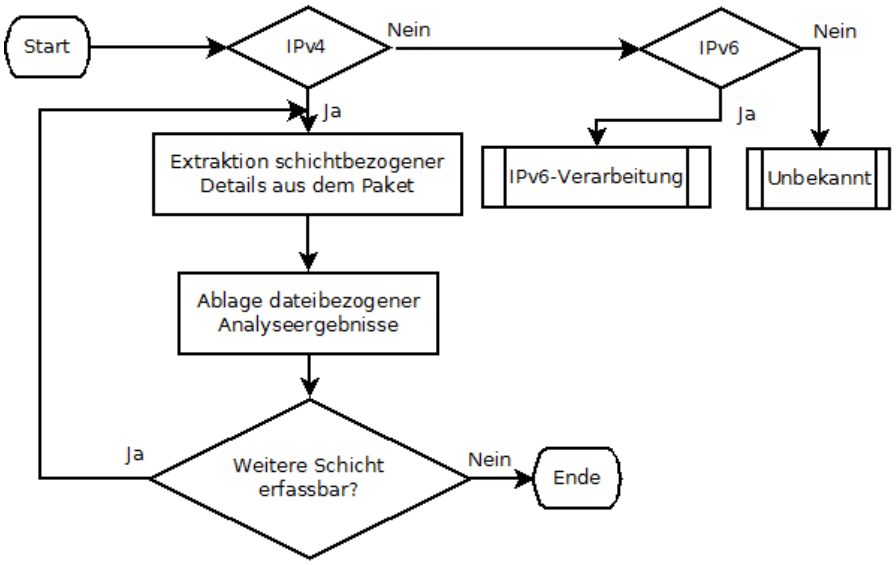

Abbildung 4.4.: Schichtweise-Analyse

endgültigen Daten auftauchen. Dazu kommen auch hier die im Kapitel 5.4 definierten Datenstrukturen für eine effizientere Speicherung zum Einsatz.

5. Implementierung

Dieses Kapitel umfasst die Beschreibung der Implementierung, wobei die verwendeten Frameworks, Sprachen und Hilfsmittel genannt und im Bedarfsfall an die geforderten Umstände angepasst werden.

Die Klassifikationssoftware wurde aufgrund der Verbreitung, Portabilität sowie der Verfügbarkeit von bestehenden Entwicklungen in der Sprache Java geschrieben. Die Grundfunktionen der PCAP-Verarbeitung stellt hierbei die Bibliothek Netutils bereit, wobei diese das Öffnen der PCAP-Dateien übernimmt und eine iterative Verarbeitung von einzelnen Ethernetrahmen anbietet. Des Weiteren sind grundlegende Funktionen zum Erkennen und Behandeln von IPv4-, TCP-, UDP- und ICMP-Paketen enthalten, die für die mehrschichtige Analyse eingesetzt werden.

Der Ablauf der Implementierung orientiert sich an der im Konzept vorgestellten Reihenfolge. Nach der Vorbereitung und Aufteilung der Daten folgt die Verarbeitung einer Datei, in deren Verlauf die mehrschichtige Analyse beschrieben wird. Anschließend folgt die Zusammensetzung der Daten aus den einzelnen Teilanalysen.

5.1. Einlesen von Dateien

Die Klassifikationssoftware ist in der Lage, ein gegebenes Verzeichnis rekursiv nach *.pcap-Dateien zu durchsuchen. Dabei wird die lexikographische Sortierung der Dateien als Ordnungskriterium für die Dateien verwendet. Während diesem Vorgang wird versucht für jede erkannte Datei die Meta-Daten aus der Datenbank abzurufen. Falls diese nicht vorhanden sind, werden diese zunächst dort abgelegt. Alle eingelesenen Dateien werden danach in einer grafischen Oberfläche aufgelistet

und können einzeln oder komplett mit einem beliebigen Verbraucher (5.2) durchlaufen werden. Die grafische Oberfläche ist dabei nur eine Entwicklungsversion, die aufgrund der Verwendung des Model-View-Controller Modells (2.4.1) mit einer endbenutzerorientierten Version ausgetauscht werden kann.

5.1.1. Speicherung von Meta-Daten

Zur Speicherung der Meta-Daten wurde aufgrund der Objektorientierung von Java eine passende objektorientierte Datenbank gewählt. MongoDB verwaltet hierbei die bereits bekannten Meta-Daten. Dabei ist die Software mit Hilfe des Spring-MongoDB-Frameworks angebunden. Durch den Einsatz des Singleton-Musters (2.4.2) in Kombination mit einer Schnittstelle, welche die notwendigen Datenbankfunktionen vorgibt, lassen sich so auch andere Datenbanken leicht an die Klassifikationssoftware anbinden. Bei der konzeptuellen Entwicklung der Software wurden so auch die Datenbanksysteme MySQL und SQLite verwendet.

Je nach Konfiguration der Klassifikationssoftware lassen sich dadurch auch einfach andere Informationen im Verlauf der Analyse speichern. Pakete, bei denen in der schichtweisen Interpretation eine nicht behandelbare Ausnahme auftritt, lassen sich so bei Bedarf für eine spätere Untersuchung auffangen. Kleinere Datenmengen, die während der Analyse anfallen, können so ebenfalls in der Datenbank anstatt in einer Datei zwischengespeichert werden.

5.2. Verarbeitung einer Datei

Sobald alle Meta-Daten in der Datenbank vorliegen, kann für jede Datei sequentiell oder parallel ein Erzeuger-Verbraucher-Paar mitsamt eigener Paketwarteschlange gestartet werden. Dabei basiert die Warteschlange auf der in Java verfügbaren Klasse LinkedBlockingQueue. Diese kann wie im Codeteil 5.2 zu sehen mit einem optionalen Parameter in der Größe beschränkt werden, wobei mit Größe die Anzahl der aufzunehmenden Objekte gemeint ist. Diese sollte sich an dem Aufwand der durchzuführenden Analyse und natürlich an der verfügbaren Speichermenge orientieren.

Der Erzeuger iteriert hierbei durch die Datei und wendet auf jedes erfolgreich erstellte Paket bereits die ersten aktivierten Filter an, bevor er versucht, dieses

```
1 LinkedBlockingQueue(int capacity)
2         Creates a LinkedBlockingQueue with the given
3         (fixed) capacity.
```

Listing 5.1: JavaDoc LinkedBlockingQueue [Cor13]

in die Warteschlange zu legen. Sollte die Warteschlange vollständig gefüllt sein, blockiert der entsprechende Erzeuger-Thread und gibt die Systemressourcen frei. Der bis dato einzige vor dem Verbrauchen angewandte Filter beinhaltet die Überprüfung, ob es sich bei dem Paket um ein ausgehendes Paket handelt und sortiert dieses dann bereits vor der Warteschlange aus. Da der überwachte Adressraum in dieser Arbeit bei 134.91.78.160 startete und bei 134.91.78.254 endete, reichte für das Ausfiltern der ausgehenden Pakete ein Vergleich der Quell-Adresse des Paketes mit dem umgerechneten Wertebereich der lokalen Adressen (siehe Zeile 9 in 5.2).

```
1  // 134.91.78.160 = 2254130848
2  long startAdress = new Long( "2254130848" );
3
4  // 134.91.78.254 = 2254130942
5  long endAdress = new Long( "2254130942" );
6
7  [...]
8
9  if ( sourceIp >= startAdress && sourceIp <= endAdress ) {
10         matchCounter++;
11         return true; // paket matches filter
12 }
```

Listing 5.2: Filtern ausgehender Pakete

An dieser Stelle lassen sich auch andere Filterkriterien wie eine zeitliche Beschränkung der durchzulassenden Pakete nachträglich realisieren. Hierbei könnten aber auch ganze Protokolle oder das im Kapitel 3.2.1 beschriebene Quell-Ziel basierte Filtern umgesetzt werden.

Für die eigentliche Analyse eines Paketes nach dem Entnehmen aus der Warteschlange bietet die Klassifikationssoftware derzeit drei verschiedene Arten von Verbrauchern an. Die spezialisierenden Verbraucher erben von der Klasse Consu-

mer, welche dabei die im Konzept gezeigten Modi implementiert und dabei die Entnahme der Pakete und die Kommunikation mit den Erzeugern übernimmt. Die abgeleiteten spezialisierten Verbraucher-Klassen müssen daher nur noch implementieren, wie ein Paket zu behandeln ist und welche Aktionen zum Sichern der Zwischenergebnisse nach Abschluss der aktuell in Verarbeitung befindlichen Datei durchzuführen sind.

1. StoreConsumer
2. TimeConsumer
3. ClassificationConsumer

Der **StoreConsumer (1)** bietet einzig und allein die Funktion, Pakete aus der Schlange zu entnehmen, diese auf den Protokoll-Typ (TCP, UDP, ICMP) zu untersuchen und anschließend entsprechende Objekte von den Paketen zu erstellen. Danach werden diese der Datenbank zur persistenten Speicherung übergeben, von der aus eine spätere Auswertung möglich ist. Dieser Consumer bietet sich allerdings nur für eine sehr kleine Anzahl Pakete an.

Der **TimeConsumer (2)** extrahiert aus jedem Paket die zeitlichen Meta-Daten. So wird zum Beispiel der Zeitstempel des ersten und des letzten Paketes einer .pcap-Datei ermittelt. Dieser Verbraucher wurde auch für die Erstellung der zeitlichen Verteilung eingehender Pakete pro Stunde, welche im Kapitel Analyse (6) zu sehen ist, verwendet.

Der primär relevante Verbraucher dieser Arbeit ist jedoch der **Classification-Consumer (3)**. Dieser beinhaltet die im Konzept vorgestellte schichtweise Interpretation der Pakete und umfasst daher auch die Identifikation der Protokolle auf den einzelnen TCP/IP-Ebenen (2.1). Abbildung 5.1 zeigt dabei die schichtweise Übertragung der gewonnenen Daten in eine entsprechende Vererbungshierarchie.

5.2.1. Schichtweise Interpretation eines Paketes

Um ein Paket als UDP, TCP oder ICMP zu klassifizieren, wird das Protokollfeld aus dem IP-Header (2.2) ausgelesen. Für die beiden erstgenannten Protokolle bietet die Netutils-Bibliothek bereits vorgefertigte Testmethoden an. Für ICMP

beinhaltet die Klassifikationssoftware auf der für die Vermittlungsschicht zuständigen Ebene eine an die Funktionen von der Netutils Bibliothek angelehnte Methode, die das Protokollfeld aus dem IPV4-Paket auf den Typ ICMP (0x1) testet.

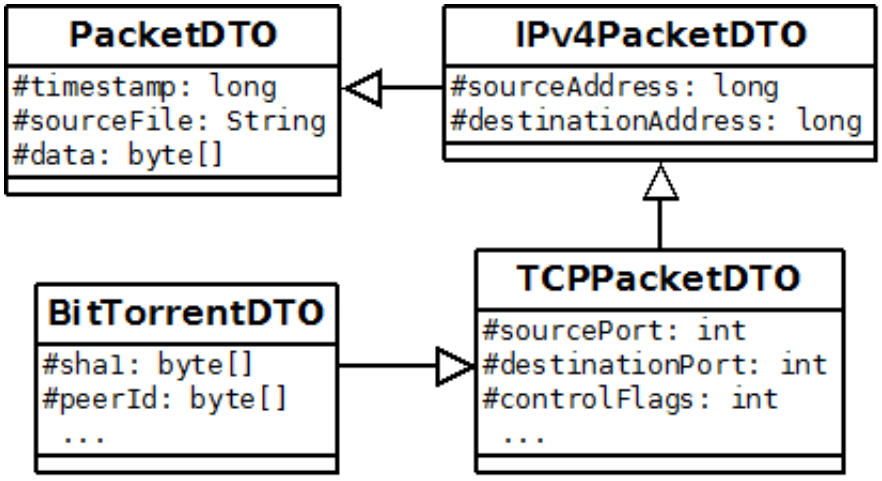

Abbildung 5.1.: Vererbungshierarchie eines als BitTorrent klassifizierten Pakets

Bei einem erfolgreich als UDP erkannten Paket mit Zielport 53 wird anschließend versucht die Nutzdaten als DNS-Paket zu interpretieren. Dazu verwendet die Klassifikationssoftware die Bibliothek dnsjava von Xbill [Wel13]. Die Nutzdaten eines vermeintlichen DNS-Paketes werden dazu an die Klasse Message der Bibliothek übergeben. Diese implementiert das DNS-Format nach RFC 1035 [Moc87], von welcher aus die für die Analyse relevanten Daten wie zum Beispiel die angefragten DNS-Querys und -Responses einfach extrahiert werden können.

Für die Behandlung von SIP-Paketen wird die Bibliothek mjsip [mjs13] verwendet. Hierbei werden vergleichbar mit der Behandlung von DNS-Paketen bei Auftreten des Zielports 5060 die Nutzdaten des UDP-Paketes an eine Schnittstelle der Bibliothek gegeben, welche daraufhin ein Objekt vom Typ einer SIP-Message zurück liefert.

Für andere Protokolle wie BitTorrent oder HTTP, welche beide auf der TCP-Ebene angesiedelt sind, verwendet die Klassifizierungssoftware derzeit eigenständige Entwicklungen zum Erkennen der relevanten Daten. An dieser Stelle lässt sich die Software an andere Analysen anpassen und um neue Protokolle erweitern, da neue Formate einfach hinzugefügt oder bestehende an aktuelle Gegebenheiten angepasst werden können.

5.3. Erfasste Daten einzelner Dateien zwischenspeichern

Um die Daten der einzeln analysierten Dateien zu speichern, sind je nach Informationsart unterschiedliche Datenstrukturen erforderlich. Dabei reichen für die Speicherung der Zwischen- und Endergebnisse zwei Technologie unabhängige Methoden, die eine Konvertierung von der Objektstruktur in Java in ein beliebiges Speicherformat gewährleisten. Die konkrete Ausgestaltung hängt hierbei von dem jeweiligen Format selber ab und muss beim Einbau einer neuen zu speichernden Objektart für diese vollständig spezifiziert und implementiert werden.

```
1 public interface SerializableEntry<E, K> {
2
3     public E deSerialize( K k );
4
5     public K serialize( );
6 }
```

Listing 5.3: Filtern ausgehender Pakete

Für eine Verwendung der später beschriebenen Datenstrukturen (5.4) muss jedes zu speichernde Objekt das im Codeteil 5.3 dargestellte Interface implementieren.

5.3.1. Comma Separated Values (CSV)

Um die Ergebnisse einer Analyseschicht persistent für eine spätere Auswertung zu Speichern reicht in vielen Fällen das einfache CSV-Format aus. Dabei übernimmt die Bibliothek opencsv ([GS11]) das entsprechende Einlesen von CSV-Dateien. Als Beispiel werden so die Anzahl der Bytes, die summiert die Größe eines TCP-Paketes anzeigen, zeilenweise in einer CSV-Datei abgelegt. Dabei besteht jede Zeile, wie im Codeteil 5.4 zu sehen, aus der Anzahl Bytes im Kopfteil, separiert von der Anzahl Bytes in den Nutzdaten. Bei der späteren Speicherung aller Zeilen wird in jeder Zeile noch die Anzahl der auftretenden Häufigkeit von identischen Einträgen aufgeführt, um mehrfaches Speichern identischer Zeilen zu vermeiden.

```
1 public String serialize() {
2         StringBuffer bf = new StringBuffer();
3         bf.append( headerSize );
4         bf.append( ';' );
5         bf.append( payloadSize );
6         return bf.toString();
7 }
8 // Ausgabe Beispiel: 20;0
9 // Beispiel mit Anz. identischer Zeilen: 20;0;6576
```

Listing 5.4: Beispiel von TCP-Packetgrößen im CSV-Format inkl. Serialisierung

5.3.2. JavaScript Object Notation (JSON)

Komplexere Datenstrukturen wie Arrays sind mit einem rein auf Trennzeichen basierendem Dateiformat nur schwer abzubilden. JSON erweitert hierbei die auf Trennzeichen basierende Speicherung um mögliche Verschachtelungen und um eine einfachen Listenstruktur. Jedes zu speichernde Objekt muss hier wiederum die Hin- und Rückkodierung selbstständig definieren, sodass die später beschriebenen Hilfsfunktionen zur Speicherung der Datenstrukturen sich nicht mehr um diesen Teil der Umwandlung kümmern müssen. Für die Umsetzung in JSON selbst wird dabei die Bibliothek json-simple [Fan09] verwendet. Hiermit können zum Beispiel DNS-Anfragen mitsamt deren eventuell enthaltenen Querys und Responses serialisiert und dateiübergreifend mit später auftretenden DNS-Anfrage verglichen werden.

5.4. Datenstrukturen

Bei der Ermittlung der Analyse-Ergebnisse gilt es verschiedene Datentypen und Informationen auf unterschiedliche Weise zu verarbeiten. Dabei gehen die Anforderungen vom reinen Zählen bis hin zu komplexen Vergleichen von ganzen Datencontainern auf Un- oder Gleichheit. Die von Java 7 angebotenen Datenstrukturen liefern in Kombination mit der Objektorientierung bereits viele hilfreiche Features. Eine fehlende Funktion wird anhand eines Beispiels im Folgenden exemplarisch beschrieben.

5.4.1. Zugriffszählende HashMap

Bei vielen auftretenden Daten wäre eine reine Speicherung der auftretenden Objekte sehr speicherintensiv, da mehrfach auftretende Werte unnötig Platz belegen würden. Um dennoch möglichst viele Informationen aus der Menge an Paketen zu extrahieren bietet es sich an mehrfach vorkommende Einträge zu gruppieren und dabei das mehrfache Vorkommen zu zählen. Die Nutzdatenlänge oder der Ziel-Port eines Paketes sind zum Beispiel auf fast jeder Schicht ein interessanter Wert, den es ressourcenschonend zu erfassen gilt. In der zugriffszählenden HashMap werden die auftretenden Paketgrößen als Schlüssel eingefügt, während der zugehörige Wert die Anzahl an bisherigen Einfüge-Versuche dargestellt. Bei jedem Versuch, einen bereits vorhandenen Schlüssel einzufügen, wird die Anzahl im Wert-Feld um eins inkrementiert.

```
1  HashMap<E, Integer> entrys;
2
3  boolean add( E e ) {
4          if ( entrys.containsKey( e ) ) {
5                  int i = entrys.get( e );
6                  i++;
7                  entrys.put( e, i );
8                  return  false;
9          } else {
10                 entrys.put( e, 1 );
11                 return true;
12         }
13 }
```

Listing 5.5: Einfügen von Objekten in eine Zugriffszählende HashMap

Beim Zusammenfügen der zwischengespeicherten Dateien aller Analyseschichten werden die auftretenden Schlüssel wiederum auf mehrfaches Auftreten überprüft und im positiven Fall die dazugehörigen Werte einfach summiert. Dank der Objektorientierung in Java können in der zugriffszählenden HashMap jedoch nicht nur einfache Zahlen, sondern auch ganze Objekte als Schlüssel eingefügt und gezählt werden. Dazu muss jede Datenstruktur die .hashCode()- und .equals()-Methode definieren. Die .containsKey()-Methode in Zeile zwei im Code 5.4.1 der zugriffszählenden HashMap ruft dabei jeweils die .hashCode()-Methode des zu speichernden Ob-

jekts auf. Diese muss sicherstellen, dass zwei gleiche Objekte im Sinne der Zählung denselben Wert als Hashwert beim Aufruf zurück liefern. Beispielhaft ist eine solche .hashCode()-Methode im Code-Block 5.4.1 zu sehen. Die dort exemplarisch gezeigte .hashCode()-Methode einer DNS-Nachricht besteht zum einem aus einem Session-Objekt, welches die Quell- und Ziel-Adresse der Nachricht beinhaltet, außerdem sind darin auch die in dieser Nachricht enthaltenen Querys und Responses enthalten. Dadurch können identische Anfragen zwischen einem Ziel-Host-Paar gezählt werden.

```
1 public int hashCode() {
2         int result = 17;
3         result = 31 * result + session.hashCode();
4         result = 31 * result + Arrays.hashCode( questions );
5         result = 31 * result + Arrays.hashCode( answers );
6         return result;
7 }
```

Listing 5.6: HashCode Methode eines DnsMessage-Objektes

5.4.2. Programmweite Hilfsfunktionen zur Speicherung

Alle Objekte, welche das weiter oben beschriebene Interface zur Serialisierung implementieren und dabei auch die für die Verarbeitung innerhalb einer HashMap benötigten Methoden überschreiben, können so in einer der zählenden Datenstrukturen abgelegt werden. Diese Datenstruktur abstrahiert dabei von ihrem Inhalt, weshalb allgemeine Methoden zum Speichern und wieder Einlesen der Daten für das spätere Zusammensetzen innerhalb der Klassifikationssoftware an zentraler Stelle implementiert sind.

5.5. Zusammensetzen der Zwischenergebnisse

Nach dem vollständigen Durchlauf eines Datensatzes können die Zwischenergebnisse einer Analyse anschließend aggregiert werden. Dazu liest die Software das temporäre Verzeichnis rekursiv aus und sucht innerhalb jeden Ordners nach Dateien mit identischem Namen. Die auf IPv4-Ebene gesammelten Quell-Adressen werden

dabei zum Beispiel mit dem Namen "IPV4_distinctSourceAdresses_counted.csv "gespeichert. Beim Aggregieren der Daten werden diese Dateien sequentiell eingelesen und die gelesenen Adressen in einer gemeinsamen Datenstrukutr abgelegt, doppelt auftauchende Adressen werden hierbei wiederum nur gezählt. Abschließend werden die nun endgültig einzigartigen Adressen in dem Verzeichnis für die endgültigen Analyseergebnisse abgelegt. Im Codeteil 5.5 wird dabei exemplarisch gezeigt, wie das zusammenfügen der Daten im Kern realisiert ist.

```
1 Given: readed entry with a key and a value;
2
3 if ( getEntrys().containsKey( key ) ) {
4         long tmp = getEntrys().get( key );
5         getEntrys().put( key, val + tmp );
6 } else {
7         getEntrys().put( key, val );
8 }
```

Listing 5.7: Einfügen der gelesenen Werte in eine HashMap

Innerhalb einer zusammenfügenden Instanz wird dabei eine HashMap mit den gelesenen Werten gefüllt, wobei vor dem Einfügen mit Hilfe der .contains()-Methode geprüft wird, ob ein identischer Datensatz bereits im Speicher abgelegt ist. Sollte dies der Fall sein, wird der dazu gehörende Zähler aus dem Speicher abgerufen und mit dem vom aktuell gelesenen Wert addiert. Anschließend wird das im Speicher befindliche Schlüssel-Wert-Paar mit den aggregierten Daten überschrieben.

6. Analyse und Klassifizierung

In diesem Abschnitt folgt die Zusammenfassung der Analyseergebnisse, welche mit Hilfe der Klassifkationssoftware aus dem Beispieldatensatz (4.1) extrahiert werden konnten. Dabei werden allgemeine statistische Daten in sortierter und nach Relevanz gekürzter Form, die Verteilung der Protokolle innerhalb der Schichten oder aber auch eine Herkunftsanalyse der Daten beschrieben.

Bei den im folgenden dargestellten Tabellen handelt es sich dabei meistens um stark gekürzte Formen, wobei zum Ausgleich unter jede Tabelle die Gesamtanzahl der betroffenen Objekte aufgeführt ist. Sofern nur eine Gesamtanzahl angegeben ist, gilt diese für beide Seiten der Tabelle.

6.1. Vorverarbeitung

Die Aufteilung des Beispieldatensatzes in kleinere Dateien wurde mit Hilfe von SplitCap [AB13b] durchgeführt. Dabei wurde die Dateigröße auf maximal 5.000.000 Pakete pro Datei begrenzt. Aus den ursprünglich sieben Dateien wurden dabei 233 kleinere Dateien.

Im Anschluss daran wurden mit Hilfe der Software tcpdump [AB13a] die ausgehenden Pakete herausgefiltert. Dieselbe Aufgabe kann jedoch auch mit Filtern innerhalb der Klassifikationssoftware durchgeführt werden. Die Vorverarbeitung mit tcpdump diente einerseits der Verkleinerung der Datenmenge, auf der anderen Seite wurden so Testdurchläufe der Klassifikationssoftware in der Entwicklungsphase beschleunigt. Die Datenmenge wurde durch das Entfernen der ausgehenden Pakete um ein Drittel, das heißt von etwa 153 GB auf rund 103 GB, verkleinert.

6.2. Analyseergebnisse und Statistiken

Der für die Entwicklung zur Verfügung stehende Beispieldatensatz enthielt ca. 1,84 Milliarden Pakete, welche durch die Filterung der ausgehenden Pakete (4.2.1) auf 1,15 Milliarden Pakete (63,33%) verringert wurden. Dabei betrug der Durchschnitt der relevanten Pakete auch in den sieben von der Größe stark unterschiedlichen Dateien in etwa 62%.

RU	810603735	70,79%
UA	133931533	11,70%
BY	89639168	7,83%
KZ	26319189	2,30%
DE	20579632	1,80%

Gesamt:	1145114468

(a) Anzahl Pakete

RU	3026712	63,37%
UA	701232	14,68%
BY	439310	9,20%
KZ	173013	3,62%
CN	51016	1,07%

Gesamt:	4776085

(b) Anzahl Quelladressen

Abbildung 6.1.: Ländercodes bezogen auf die Anzahl Pakete / Quelladresse

Der erste Zeitstempel innerhalb der Beispieldaten verweist auf den 16. November 2012 und markiert damit den Beginn der Datenaufzeichnung. Der letzte Zeitstempel innerhalb des Beispieldatensatzes ist der 12. Juni 2013. In Abbildung 6.2 ist dazu passend die durchschnittliche Anzahl der eingehende Pakete pro Stunde zu sehen.

Davon abweichend stammen jedoch die meisten Anfragen mit etwas mehr als 70% aus Russland, worauf mit einigem Abstand, aber noch weit vor Paketen und Quelladressen aus Deutschland, weitere östliche Länder wie die Ukraine, Weißrussland und Kasachstan folgen. Hierbei wurden die Quell-Adressen der Pakete innerhalb einer zugriffszählenden HashMap (5.4) gespeichert. Zu jeder Adresse wurden danach die Anzahl der eingehenden Pakete gezählt und anschließend mit Hilfe einer externen Geo-IP Datenbank das voraussichtliche Herkunftsland ermittelt.

6.3. Vermittlungsschicht

Innerhalb der verarbeiteten Daten kamen keine IPv6-Daten direkt vor, es konnten jedoch 10.794 (0,001%) als in IPv4 getunnelte Pakete in Form des Teredo-Protokolls identifiziert werden. Außerdem konnte innerhalb der Nutzdaten eines

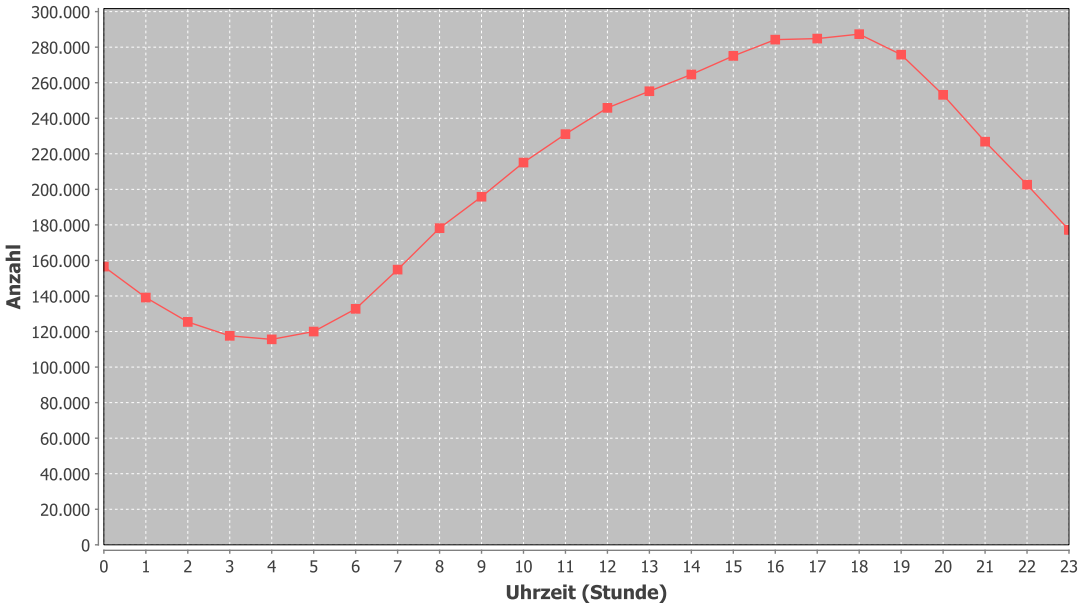

Abbildung 6.2.: Zeitliche Verteilung eingehender Pakete

ICMP-Paketes ein Paket identifiziert werden, dass dem IPv6- Protokoll entspricht. Hierbei handelte es sich offenbar nicht um Teredo und der Fehlercode im ICMP-Teil zeigte dabei an, dass für eine erfolgreiche Übertragung eine Fragmentierung notwendig ist. Aufgrund der Unvollständigkeit waren keine weiteren Informationen zu erkennen.

Die zu Beginn des Kapitels erwähnten rund 1,15 Milliarden entsprachen mit Ausnahme von acht nicht identifizierten Paketen somit alle dem IPv4-Protokoll. Während der Inhaltsanalysen innerhalb der höheren Schichten sind im weiteren Verlauf bei 0,004% nicht automatisch behandelbare Ausnahmen aufgetreten. Diese Ausnahmen stammen dabei entweder von Angriffen in Form von provozierten Pufferüberläufen oder Denial of Service Attacken, andererseits können aber auch innerhalb der schichtweisen Analysen Fehler auftreten, falls spezielle Fälle einzelner Protokolle nicht vom betroffenen Klassifizierer unterstüzt werden.

Fragmentierung

Innerhalb der ersten Testdurchläufe, bei welchen auch die Anzahl der vorhandenen Pakete zur Abschätzung des Aufwandes gezählt wurden, konnte im Falle eines IP-Paketes bereits das *MF-Bit* der Controlflags ausgelesen werden. Dabei wurde die

Funktion des Zusammensetzen von fragmentierten Paketen als überflüssig deklariert, da kein gesetztes *MF-Bit* und somit keine fragmentierten Pakete innerhalb der Daten vorlagen.

ICMP

Mit rund 800.000 Paketen und damit nur 0,07% waren ICMP-Pakete innerhalb der Daten zu finden. In Abbildung 6.3 ist dabei zu sehen, dass sich der Typ hierbei auf die drei Arten „Ziel nicht erreichbar“, „Echo“ und „Zeit überschritten“ beschränkt und jeweils mit unterschiedlichen ICMP-Codes kombiniert ausgeliefert wurde.

TYP		CODE		Anzahl	Prozent
3	Destination Unreachable	1	Host Unreachable	329693	40,62%
11	Time Exceeded	0	TTL exceeded in Transit	248338	30,60%
8	Echo	0	No Code	91973	11,33%
3	Destination Unreachable	13	Communication AP*	46177	5,69%
3	Destination Unreachable	0	Net Unreachable	44461	5,48%
3	Destination Unreachable	10	Communication with Destination Host is AP*	28352	3,49%
3	Destination Unreachable	3	Port Unreachable	20615	2,54%
0	Echo Reply	0	No Code	1279	0,16%
3	Destination Unreachable	2	Protocol Unreachable	441	0,05%
3	Destination Unreachable	4	Fragmentation Needed and DF was Set	128	0,02%
3	Destination Unreachable	9	Communication with Destination Network is AP*	78	0,01%
8	Echo	9	?	47	0,01%
11	Time Exceeded	1	Fragment Reassembly Time Exceeded	2	0,0002%

*Administratively Prohibited

Summe: 811584

Abbildung 6.3.: Arten von Typen und Codes, ICMP

6.4. Transportschicht

Innerhalb der Transportschicht stellt TCP mit über 99,55% die große Mehrheit im Vergleich zu UDP mit 0,38% dar. Die restlichen 0,07% sind dabei die eben beschriebenen ICMP-Daten (6.3). ICMP stellt zwar ein eigenständiges Protokoll dar, dieses wird innerhalb der Klassifikationssoftware ebenfalls auf der Transportschicht des TCP/IP-Modells (2.1) erfasst.

6.4.1. TCP

Das TCP-Protokoll hat im Vergleich zu anderen Protokollen den Nachteil, dass einige Vorteile, wie die sichere Zustellung von Paketen und andere positive Eigenschaften, mit einer Reihe an zusätzlichen Kontroll- und Steuerungsdaten erkauft werden. Alleine der Auf- und Abbau einer Sitzung und auch andere Funktionen wie die Abweisung aufgrund geschlossener Ports (2.2.3) erzeugen viele zusätzliche Pakete, die übertragen werden.

Kopfteil	Nutzdaten	Anzahl	%	Kopfteil	Nutzdaten	Anzahl	%
20	0	473106419	41,50%	36	0	1224101	0,11%
20	68	284402407	24,95%	24	0	582846	0,05%
32	0	257316355	22,57%	20	1	308119	0,03%
28	0	74672749	6,55%	20	31	305609	0,03%
40	0	18798787	1,65%	20	11	241211	0,02%
32	68	15181474	1,33%	20	52	194773	0,02%
20	22	3075219	0,27%	32	41	117226	0,01%
44	0	1309745	0,11%	32	22	86806	0,01%

Gesamt: 1139992686

Abbildung 6.4.: Anzahl der Bytes von Kopf- und Nutzdaten, TCP

Dies konnte auch in den Beispieldaten nachgewiesen werden, da gut 72% der eingehenden TCP-Pakete, also ein Großteil der überhaupt eingehenden Daten, Anfragen ohne Nutzdaten waren. Die Pakete, die nur aus den 20 Byte des Kopfteiles bestehen, resultieren dabei oftmals von automatisierten Abfragen ganzer Port-Bereiche am Zielsystem. Anfragen mit 20 bzw. 32 Byte im Kopfteil und 68 Byte Nutzdaten stellen dabei nahezu exakt die später beschriebenen 26% der erkannten BitTorrent-Pakete (6.5) aus der Anwendungsschicht dar. Von den etwas mehr als 27% der TCP-Verbindungen mit Nutzdaten konnten also ein Großteil der Inhalte erfolgreich identifiziert werden. In der Tabelle 6.5 sind die angefragten TCP-Controlflags aufgeführt, wobei auf der rechten Seite die anfragten Flags hervorgehoben sind. Mit 58 vorkommenden Flag-Kombinationen kamen dabei fast alle der maximal Möglichen 64 Kombinationen vor, obwohl nicht alle davon eine gültige Kombination im TCP-Protokoll darstellen. Die meisten Anfragen hatten hierbei ein gesetztes ACK- und ein ebenfalls gesetztes PSH-Bit. Die reinen Verbindungsanfragen liegen dabei mit rund 25% an dritter Stelle. Daraufhin folgen die Verbindungen, die durch ein gesetztes RST-Bit einen Fehler oder eine Abweisung der Verbindung anzeigen.

Flags	Flags in BIN	Anzahl		URG	ACK	PSH	RST	SYN	FIN
24	011000	309467995	27,15%	0	1	1	0	0	0
16	010000	290422784	25,48%	0	1	0	0	0	0
2	000010	285591380	25,05%	0	0	0	0	1	0
20	010100	229198390	20,11%	0	1	0	1	0	0
17	010001	16795285	1,47%	0	1	0	0	0	1
4	000100	8139134	0,71%	0	0	0	1	0	0

Gesamt:	1139992686

Abbildung 6.5.: Vorkommen der Control-Flags, TCP

Protokoll	Port	Anzahl	%
RDP	3389	2607335	0,23%
VNC	5900	1549777	0,14%
MS-SQL	1433	1083096	0,10%
SSH	22	998418	0,09%
HTTP	80	579164	0,05%
TELNET	23	411655	0,04%
HTTP	8080	201003	0,02%
SSL	443	136059	0,01%
POP3	110	56405	0,005%
FTP	21	19259	0,002%

Port	Anzahl	%
60096	115144070	10,10%
61978	95773525	8,40%
6901	77345156	6,78%
48859	73005974	6,40%
59162	66768116	5,86%
20250	58625425	5,14%
17690	54355877	4,77%
28186	53097452	4,66%
3080	49377794	4,33%
25370	39436116	3,46%

Gesamt:	1139992686

Abbildung 6.6.: Top 10 der Well-Known-Ports, Top 10 der angefragten Ports, TCP

Die Verteilung der Anfragen auf die unterschiedlichen Ports ist in der Tabelle 6.6 zu sehen. Auf der linken Seite sind dabei die zehn meist angefragten Well-Known Ports zu sehen, wobei hier bisher keine weitere Prüfung der Daten vorgenommen wurde. Auf der rechten Seite sind die zehn meist angefragten Ports zu erkennen. Bei dem Versuch diese zu identifizieren, wurde festgestellt, dass diese mit den Ports bei erkannten BitTorrent-Anfragen (A.1) übereinstimmen. Diese sind für einen Vergleich dafür im Anhang A.1 dargestellt.

6.4.2. UDP

Innerhalb von UDP kamen die bereits erwähnten 0,001% von getunnelten IPv6-Pakete vor. Anders als bei TCP, waren hier keine eingehenden Anfragen ohne Nutzdaten enthalten. Die auftretenden Nutzdatenlängen sind in der Tabelle 6.7 zu sehen.

Ziel-Port	Anzahl	%
8611	718983	16,68%
48457	316477	7,34%
20250	297141	6,89%
35610	234888	5,45%
14106	144434	3,35%
58394	139485	3,24%
59162	135650	3,15%
5402	116137	2,69%
28186	109263	2,54%
59418	108906	2,53%

Gesamt:	4309827

Ziel-Port	Anzahl	%	Protokoll
53	45637	1,06%	DNS
137	1151	0,03%	NetBIOS

Größe	Anzahl	%
30	1296917	30,09%
20	978613	22,71%
16	742410	17,23%
65	527911	12,25%
67	368662	8,55%
33	168400	3,91%

Gesamt:	4309827

Abbildung 6.7.: Zielports und Nutzdatengröße in Byte, UDP

Die erste Zeile der Paketgrößen mit den ständig auftretenden 30 Bytes ist hierbei aller Wahrscheinlichkeit nach ebenfalls eine Form von Anfragen über das BitTorrent-Protokoll, was weiter im Abschnitt BitTorrent (6.5) ausgeführt wird. Die zweite Zeile, bei der die eingehende Nutzdatenlänge 20 Byte betrug, konnte dabei Teilweise als DNS (0,8%) oder aber selten auch als NTP (Network Time Protocol, Port 123) identifiziert werden.

Die in Abbildung 6.7 gezeigte erste Zeile der angezeigten Zielports mit dem Port 8611 waren dabei zumindest zu 98% Anfragen über das BJNP Protokoll von Quel-

ladressen aus dem Bereich 84.60.0.0/16 mit dem Inhalt [42, 4a, 4e, 50, 01, 01, 00, 00, XX, XX und 6 folgenden 00-Bytes]. Die Platzhalter XX stehen dabei für die Sequenznummer innerhalb des Protokolls. Diese Anfrage beruhen laut dem Analysetool WireShark und anderen Portdatenbanken wie der Sans [Cen06] auf einer Druckersoftware der Firma Canon, welche über dieses Protokoll nach Netzwerkdruckern sucht.

1,38% der UDP-Pakete die auf Port 5060 eingegangen sind, konnten unabhängig davon mit Hilfe der mjsip-Bibliothek als [mjs13] SIP-Nachricht identifiziert werden.

6.5. Anwendungsschicht

Auf der Anwendungsschicht konnten bisher Daten aus den Protokollen HTTP, DNS, SIP und BitTorrent identifiziert werden. Die daraus gewonnen Daten werden im Folgenden teilweise exemplarisch aufgezeigt, teilweise befinden sich hierfür Tabellen aufgrund Ihrer Größe im Anhang dieser Arbeit.

BitTorrent

Innerhalb von TCP sind von den rund 28% der Anfragen, die überhaupt Nutzdaten enthielten, 26,28% der Pakete als BitTorrent-Pakete erkannt worden. Da diese sich nicht wie viele andere Protokolle anhand des Ziel-Ports identifizieren lassen, sind die Nutzdaten eines von allen anderen auf TCP-Ebene befindlichen Schichten nicht identifizierbaren Pakets mit dem Standard BitTorrent-Header (A.1) verglichen worden. Daran anschließend wurde die Nutzdatenlänge des Paketes noch auf die Größe 68 Byte getestet, bevor die Extraktion der Daten für BitTorrent durchgeführt wurde.

Dabei sind die im Anhang (A.1) gezeigten Zielports die entsprechend überhaupt bei TCP dominierenden Ports (vgl. Abbildung 6.6) . Wie im Anhang A.1.1 zu sehen, enthielten gut 95% der Anfragen diesselbe PeerId und die gleichen ReserverdExtensionBytes. Die 1733 unterschiedlichen Sha1-Hashes zeigen jedoch, dass dennoch viele unterschiedliche Dateien abgefragt wurden. Die Werte innerhalb der Tabellen sind dabei die Daten aus den Byte-Werten umgerechnet in den Long-Datentyp von Java.

BitTorrent Daten konnten später aber auch bei UDP klassifiziert werden. Nach den ersten Analysedurchläufen der gesamten Daten sind bei der Auswertung der Nutzdatenlängen der UDP-Pakete aufgefallen, dass die größte Masse der Daten eine Nutzdatenlänge von 30 Byte (6.7) umfasste. Durch weitere Analysedurchläufe mit später verfeinerten Klassifizierern konnte ermittelt werden, dass es sich bei diesen Paketen höchstwahrscheinlich ebenfalls um Anfragen mit dem BitTorrent-Protokoll handelt. Innerhalb des Wikis der Software libprotoident [Gro12], welche versucht, Netzwerkdaten anhand der ersten vier Bytes zu klassifizieren, sind dabei die auf die in den Daten passenden Klassifizierungseigenschaften aufgeführt. Demnach sind diese Anfragen wohl Teil von einem verteilten BitTorrent Protokoll, welches mit Hilfe von verteilten Hash-Tabellen versucht, ohne einen zentralen Tracker auszukommen. Die eingehenden Pakete haben dabei ein festes Format, bei denen die ersten zwei Bytes zusammen mit der Nutzdatenlänge den vermutlichen Anfragetyp definieren. Die Anzahl und Art der in den Daten dazu aufgetretenen ersten beiden Bytes und die entsprechende Nutzdatenlänge sind dabei in Abbildung 6.8 aufgeführt. Insgesamt entspricht die Anzahl der Anfragen dieser Art 51,9% der überhaupt eingehenden UPD-Pakete.

Integer	Anzahl	%	Byte 1	Byte 2	Länge
577	1221971	54,63%	0x41	0x02	30
65	793238	35,46%	0x41	0x00	20
17	149094	6,67%	0x11	0x00	20
545	36390	1,63%	0x21	0x02	30
33	31672	1,42%	0x21	0x00	20
49	4418	0,20%	0x31	0x00	20

Gesamt:	2236783

Abbildung 6.8.: Vermutliche BitTorrent-Anfragen, UDP

HTTP

Innerhalb der Daten konnte bei etwas weniger als 90.000 Anfragen erfolgreich das HTTP Protokoll auf Port 80 oder 8080 extrahiert werden. Dabei sind wie in Abbildung 6.9 zu sehen, 72,43% der Anfragen vom Typ GET. Die angefragten URLs und

Nummer	Anzahl	%	Methode
1	64601	72,43%	GET
2	23124	25,93%	HEAD
7	1282	1,44%	CONNECT
3	138	0,15%	POST
0	40	0,04%	OPTIONS
4	1	0,001%	PUT

Gesamt:	89186

Abbildung 6.9.: Angefragte Methode, HTTP

User-Agents sind dabei im Anhang (A.1.2) aufgeführt. Bei diesen lässt sich erkennen, dass viele Anfragen versuchen, Installationen der Blogsoftware Word-Press zu lokalisieren. Aufgrund der hohen Verbreitung von Wordpress und vieler bekannten Schwachstellen, ist diese für die Aufnahme in Botnetze interessant. Auch die weit verbreitete Webapplikation phpmyadmin, mit der sich Datenbanken über den Browser verwalten lassen, wird oft versucht aufzurufen. Insgesamt verteilen sich die eingehende Anfragen jedoch auf viele unterschiedliche Adressen. Viele Anfragen enthielten jedoch nicht alle Angaben, waren leer oder unvollständig. Alleine 46.596 der Anfragen enthielten zum Beispiel keine Angabe über den User-Agent.

Die häufige Abfrage von www.google.com deutet darauf hin, dass hier versucht wurde falsch konfigurierte Webserver zu entdecken, die dann als Proxy-Server fungieren, um die IP-Adressen der eigentlichen Absender vor dem eigentlichen Zielserver zu verbergen.

7. Fazit

Software, die versucht das Hintergrundrauschen im Internet zu klassifizieren, lässt sich in zwei Bereiche einteilen. Einerseits sind hier aktive Systeme zu erwähnen, die versuchen die Masse der eingehenden Daten zu reduzieren um diese zu speichern. Auf der anderen Seite steht Software die versucht, erfasste Daten zu verarbeiten und zu interpretieren. (vgl. [WKB+10, S.2])

Die in dieser Arbeit entwickelte Software ist dabei ausschließlich in die zweite Kategorie einzuordnen und zielt darauf ab, wie auch das in der Praxis häufig eingesetzte Netzwerkanalyse-Tool Wireshark, Netzwerkadministratoren und Forscher in diesem Gebiet bei Ihrer Arbeit zu unterstützen. Dazu wurde mit Hilfe eines Beispieldatensatzes (4.1) eine Klassifikationsumgebung für die Analyse von Internetverkehrsdaten entwickelt, mit welcher der hinter nicht vergebene IP-Adressen anfallende Netzwerkverkehr effizient untersucht werden kann. Ein kompletter Analysedurchlauf mit allen in dieser Arbeit aufgeführten Protokollanalysen benötigte dabei für die vorgefilterten Beispieldaten (103 GB) eine Durchlaufzeit von weniger als drei Stunden. Innerhalb kurzer Zeit ließen sich danach die dort gesammelten Meta-Daten wie die Paketnutzdatengröße und andere Faktoren auswerten, worauf dann zu den Daten passende Anwendungsschicht-Klassifizierer eingebaut werden konnten. Dies ist unter anderem im Kapitel Analyse (6.4.2) am Beispiel der eingehenden UDP-Pakete mit einer Nutzdatengröße von 30 Byte so durchgeführt worden.

Das Ziel dieser Arbeit, ein grundlegendes Konzept (4) zur Analyse und Klassifikation des Hintergrundrauschens im Internet sowie eine lauffähige Analysesoftware 5 zu entwickeln, ist dabei erreicht worden. Diese kann dabei an den für die Analyse relevanten Stellen einfach an neue Gegebenheiten wie neue Protokolle oder Verfahren angepasst werden. Der eingehende Netzwerkverkehr auf nicht vergebenen IP-Adressen ist dabei noch immer eine sich ständig verändernde und keinesfalls marginale Menge an Netzwerkdaten (6), die regelmäßig analysiert werden sollte.

Die große Masse stellen dabei im aktuellen Datensatz TCP-SYN Anfragen dar, während die Masse der Pakete mit Nutzdaten im TCP- wie auch UDP-Protokoll dem BitTorrent-Protokoll zuzuordnen sind. Den in der ursprünglichen Charakterisierung [PYB+04] des Hintergrundrauschens genannten Protokollen wie zum Beispiel NetBIOS ist dabei heute nur noch eine untergeordnete Rolle zuzuschreiben.

Einschränkend muss hierbei jedoch erwähnt werden, dass der im Beispieldatensatz überwachte Adressraum nur einen Bruchteil der Größe von denen der bisherigen Ansätze aufweist. In dieser Arbeit konnte jedoch gezeigt werden, dass die Analyse und Klassifikation von Internetverkehrsdaten innerhalb eines kleinen Adressbereiches möglich ist und dabei vergleichbare Ergebnisse aufweist. Das Problem der immer knapper werdenden IPv4-Adressen ist daher für die Analyse und Klassifikation von Hintergrundrauschen im Hintergrund kein Ausschlusskriterium.

Für die Weiterentwicklung der Klassifikations- und Analyseansätze in dieser Arbeit gilt es weitere Datensätze zu analysieren und aufgrund den daraus gewonnen Erkenntnisse die Klassifizierer gegebenfalls zu modifizieren oder auch zu erweitern. Mit dem Konzept des aktiven Antwortens könnten diese kombiniert und verbessert werden, wodurch auch in weiteren Protokollen Angriffsversuche klassifiziert werden könnten.

Selbstständigkeitserklärung

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Duisburg, 9. September 2013 _____

Unterschrift

Anhang A.

Anhang

A.1. Analyse und Klassifizierung

A.1.1. BitTorrent

```
1 0x13, 0x42, 0x69, 0x74, 0x54, 0x6f, 0x72, 0x72, 0x65, 0x6e,
2 0x74, 0x20, 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x63, 0x6f, 0x6c
```

Listing A.1: Die ersten 20 Byte eines BitTorrent-Paketes

Ziel-Port	Anzahl	Anzahl (%)
60096	31361427	10,47%
61978	25638704	8,56%
6901	21021322	7,02%
48859	18756562	6,26%
59162	17867227	5,96%
20250	16181312	5,40%
28186	14762788	4,93%
17690	14558282	4,86%
3080	13188364	4,40%
25370	12304018	4,11%

Gesamt:	299645983

Abbildung A.1.: Top 10 Zielports bei BitTorrent, TCP

A.1.2. HTTP

Sha1-Hash	Anzahl	%
7021534430479790000	75440188	25,18%
-8132786760377730000	69227056	23,11%
1353195292867730000	40912423	13,66%
-4762443009430380000	26117305	8,72%
4787055189664450000	17576048	5,87%
-3625288031656020000	14366413	4,80%
5091806415332410000	12650981	4,22%
1274035533653920000	9649805	3,22%
-338235196017079000	8252994	2,75%
3825195727289360000	6689546	2,23%

Gesamt:	299593282

Abbildung A.2.: Sha1s BitTorrent, TCP

PeerID	Anzahl	%
3267158327342730000	285823744	95,40%
3261808107737910000	3632251	1,21%
3273118779826580000	2797690	0,93%
3261808107737980000	2531761	0,85%
3264343585821960000	2227839	0,74%
3264343585810030000	735651	0,25%
3264343585809900000	205232	0,07%

Gesamt:	299593282

Abbildung A.3.: PeerIds BitTorrent, TCP

ReservedExtensionByte	Anzahl	%
288256764430778000	285825789	95,40%
360314358468706000	10249377	3,42%
17592186044416	2874882	0,96%
360305562375684000	414095	0,14%
360308860910567000	222597	0,07%

Gesamt:	299593282

Abbildung A.4.: ReservedExtensionBytes BitTorrent, TCP

URL	Anzahl	%
/	5444	6,10%
/manager/status	4638	5,20%
/w00tw00t.at.ISC.SANS.DFind:)	4298	4,82%
/manager/html	4024	4,51%
http://www.google.com/	2136	2,39%
http://www.proxy-alert.com/remote.php	1407	1,58%
null	1322	1,48%
/user/soapCaller.bs	1258	1,41%
http://gameframe.net/headers	1043	1,17%
/wp//wp-login.php	953	1,07%
//wp-login.php	951	1,07%
/wordpress//wp-login.php	950	1,07%
/blog//wp-login.php	949	1,06%
/pma/scripts/setup.php	853	0,96%
/phpmyadmin/scripts/setup.php	843	0,95%
/phpMyAdmin/scripts/setup.php	833	0,93%
/w00tw00t.at.blackhats.romanian.anti-sec:)	822	0,92%
/w00tw00t.at.ISC.SANS.test0:)	760	0,85%

Gesamt:	89186

Abbildung A.5.: Angefragte URLs, HTTP

User-Agent	Anzahl	%
null (nicht angegeben)	46596	52,25%
ZmEu	5174	5,80%
Mozilla/5.0 (X11; U; Linux i686; pt-BR; rv:1.9.0.15) Gecko/[…]*	3803	4,26%
Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)	3757	4,21%
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/535.1 (KHTML, […]*	3691	4,14%
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)	2136	2,39%
Wget/1.10.2 (Red Hat modified)	2085	2,34%
Mozilla/3.0 (compatible; Indy Library)	1969	2,21%
Morfeus strikes again.	1552	1,74%
Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)	1328	1,49%
Mozilla/5.0 (Windows NT 5.1; rv:5.0) Gecko/20100101 Firef[…]*	1280	1,44%
Morfeus Fucking Scanner	1258	1,41%
Java/1.6.0_24	1238	1,39%
Toata dragostea mea pentru diavola	1235	1,38%
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.2;)	1128	1,26%
Mozilla/5.0 (Windows NT 5.2; rv:15.0) Gecko/20100101 Fire[…]*	928	1,04%
Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows NT […]*	886	0,99%
Java/1.6.0_07	837	0,94%
Opera/9.80 (Windows NT 6.1; WOW64) Presto/2.12.3[…]*	760	0,85%
Java/1.6.0_32	575	0,64%

Gesamt:	89186

Abbildung A.6.: Anfragender User-Agent, HTTP

Literaturverzeichnis

[AB88] Agency, Defense Advanced Research Projects und Internet Activities Board: *IAB official protocol standards*. RFC 1083 (Historic), Dezember 1988. Obsoleted by RFC 1100.

[AB13a] AB, NETRESEC: *tcpdump 4.4.0*. `http://www.tcpdump.org/`, 2008-2013. [Online; accessed 04.09.2013].

[AB13b] AB, NETRESEC: *splitcap 2.0*. `http://www.netresec.com/?page=SplitCap`, 2013. [Online; accessed 04.09.2013].

[Cen06] Center, Internet Strom: *Port Details: Port 8611*. `https://isc.sans.edu/port.html?port=8611`, 2006. [Online; accessed 07.09.2013].

[Cor13] Corporation, Oracle: *LinkedBlockingQueue (Java Platform SE 6)*. `http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/LinkedBlockingQueue.html`, 2013. [Online; accessed 01.09.2013].

[Fan09] Fang, Yidong: *json-simple 1.1.1*. `https://code.google.com/p/json-simple/`, 2009. [Online; accessed 04.09.2013].

[Goo13] Google, Inc.: *IPv6 – Google*. `http://www.google.de/ipv6/statistics.html`, 2013. [Online; accessed 06.08.2013].

[Gro12] Group, Wand Network Research: *libprotoident 2.0.6*. `https://secure.wand.net.nz/trac/libprotoident/wiki`, 2012. [Online; accessed 07.09.2013].

[GS11] Glen Smith, Sean Sullivan, Scott Conway: *opencsv 2.0*. `http://opencsv.sourceforge.net/`, 2011. [Online; accessed 04.09.2013].

[mjs13] mjsip.org: *mjsip 1.7*. `http://www.mjsip.org/`, 2013. [Online; accessed 04.09.2013].

[Moc87] Mockapetris, P.V.: *Domain names - implementation and specification*. RFC 1035 (INTERNET STANDARD), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604.

[Pos80] Postel, J.: *User Datagram Protocol*. RFC 768 (INTERNET STANDARD), August 1980.

[Pos81a] Postel, J.: *Internet Control Message Protocol*. RFC 792 (INTERNET STANDARD), September 1981. Updated by RFCs 950, 4884, 6633, 6918.

[Pos81b] Postel, J.: *Internet Protocol*. RFC 791 (INTERNET STANDARD), September 1981. Updated by RFCs 1349, 2474, 6864.

[Pos81c] Postel, J.: *Transmission Control Protocol*. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[PYB+04] Pang, Ruoming, Vinod Yegneswaran, Paul Barford, Vern Paxson und Larry Peterson: *Characteristics of internet background radiation*. In: *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, Seiten 27–40, New York, NY, USA, 2004. ACM.

[TW12] Tanenbaum, A.S. und D.J. Wetherall: *Computernetzwerke*. Pearson Studium - IT. Pearson Studium, 2012.

[Wel13] Wellington, Brian: *dnsjava 2.1.5*. `http://www.xbill.org/dnsjava/`, 2013. [Online; accessed 04.09.2013].

[WKB+10] Wustrow, Eric, Manish Karir, Michael Bailey, Farnam Jahanian und Geoff Huston: *Internet background radiation revisited*. In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, Seiten 62–74, New York, NY, USA, 2010. ACM.