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

Masterarbeit

Klassifikation und Auswertung semi-aktiver Messungen des Internetdatenverkehrs

Jan-Frederik Zaeske
Matrikelnummer: 3022462

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

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

18. August 2017

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

Zusammenfassung

Passive Messungen des Internet-Datenverkehrs durch große ungenutzte Adressbereiche, sogenannte *Darknets*, sind ein gängiges Mittel zur Erforschung von Phänomenen der Internet-Hintergrundstrahlung wie Scanner Aktivitäten oder Backscatter. Durch die Erweiterung solcher Messungen mittels einer Antwort-Komponente für TCP-Verbindungsanfragen lassen sich weitere Informationen zur Hintergrundstrahlung sammeln. Dies erfordert jedoch auch angepasste Verfahren zur Analyse der gesammelten Daten, da die Annahmen eines rein passiven Darknets für diese Messungen nicht mehr zutreffen.

In dieser Arbeit werden bestehende Analyseverfahren für Messungen des Internet-Datenverkehrs systematisiert und die Auswirkungen des gewählten Messverfahrens auf die Analysemethode aufgezeigt. Weiterhin trägt diese Arbeit ein iteratives Analyseverfahren bei, welches die Auswirkungen von aktiven Antwortkomponenten auf die Messung berücksichtigt. Ein weiterer Beitrag ist die Konzeption einer Analysesoftware für Internet-Datenverkehr mit Fokus auf Erweiterbarkeit und effiziente Verarbeitung großer Datenmengen. Die hierzu vorgenommenen Optimierungen ermöglichen eine um bis zu 65% schnellere Verarbeitung von PCAP-Dateien als die Standardimplementierung `libpcap`.
Aus der vorgenommenen Analyse eines mehrere Jahre umfassenden Datensatzes geht unter anderem hervor, dass eigentlich mittels *Message Stream Encryption (MSE)* verschleierter BitTorrent-Datenverkehr durch eine Häufigkeitsanalyse im Nachhinein identifiziert werden kann. Ebenso wird gezeigt, dass das Beantworten von TCP-Verbindungsanfragen zusätzliche Auswirkungen auf den eingehenden ICMP-Datenverkehr hat.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

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

Duisburg, 18. August 2017 _____

Unterschrift

Inhaltsverzeichnis

1. Motivation

Im Umfeld der Erforschung von Sicherheitsrisiken im Internet kommen unter anderem Messungen des übertragenen Datenverkehrs zum Einsatz. Dieser gibt Aufschluss darüber, welche Computer untereinander kommunizieren und welcher Art diese Kommunikation ist. Dadurch lassen sich Ursprung und Ziel von Angriffen erkennen. Eine Analyse der übertragenen Daten kann weiterhin Aufschluss über die Struktur einer Sicherheitslücke geben. Von besonderem Interesse ist dabei Datenverkehr, welcher nicht der normalen Nutzung entspricht, etwa Angriffe auf bestimmte Dienste oder der Versuch eine Sicherheitslücke in einem System auszunutzen. Diese Daten müssen zunächst aus dem regulären Datenverkehr heraus gefiltert werden. Eine Möglichkeit dieser Filterung ist der Einsatz einer Messmethode, die unter anderem als *Darknet* bezeichnet wird[1]. Dabei werden zur Messung IP-Adressen gewählt, welche sonst nicht für reguläre Kommunikation genutzt werden. Alle Daten, die dennoch an diese Adressen gesendet werden, sind aufgrund von Abweichungen zum regulären Datenverkehr entstanden. So lässt sich durch die Messmethode eine Vorfilterung vornehmen.

Die Nutzung mancher Protokolle, wie dem häufig eingesetzten Transportprotokoll TCP, erfordert, dass beide Kommunikationsteilnehmer zunächst Informationen zur Synchronisation der Verbindung senden. Erst danach werden die eigentlichen Nutzdaten übertragen werden. *Darknet*-Messungen können um eine Antwortkomponente ergänzt werden, welche eingehende Verbindungsanfragen beantwortet und somit die Übertragung dieser Daten ermöglicht. Diese hybride Art der Messung, bei der sich der Sensor grundsätzlich passiv verhält, jedoch aktiv auf eingehende Anfragen antwortet, wird im Rahmen dieser Arbeit als semi-aktive Messung bezeichnet. Im Gegensatz zu einer *Darknet*-Messung interagiert der Messsensor mit dem übrigen Internet. Diese Interaktion muss bei der Analyse der aufgezeichneten Daten berücksichtigt werden.

In dieser Arbeit wird die Aufzeichnung einer mehrjährigen semi-aktiven Messung analysiert und die Daten klassifiziert. Hierzu wird ein iteratives Analyseverfahren vorgestellt,

[1] Der Begriff wird mit abweichender Bedeutung auch im Umfeld von Anonymisierungsdiensten wie dem TOR-Netzwerk `https://www.torproject.org/` verwendet. In dieser Arbeit bezieht sich dieser Begriff auf die Messmethode.

welches die Auswirkungen der Antwortkomponente auf die Messergebnisse berücksichtigt. Dabei soll insbesondere der übertragene Payload betrachtet werden. Ebenso wird eine Analysesoftware konzipiert und umgesetzt, mit welcher sich das beschriebene Verfahren durchführen lässt. Insbesondere eine effiziente Auswertung großer Datenmengen, wie sie bei der Messung entstanden sind, wird hierbei berücksichtigt. Bei der Auswertung der Messdaten soll weiterhin untersucht werden, für welche Protokolle der Einsatz von weiteren Antwortkomponenten auf Ebene des Anwendungsprotokolls sinnvoll ist.

Dazu werden zunächst in Kapitel 2 die Grundlagen vorkommender Netzwerk-Protokolle sowie die Unterschiede zwischen verschiedenen Verfahren zur Messung von Internet-Datenverkehr beschrieben. In Kapitel 3 werden anschließend vorherige Forschungsarbeiten betrachtet, welche Internet-Datenverkehr gemessen und klassifiziert haben. Dabei werden insbesondere die Unterschiede in der Messmethodik und die daraus resultierenden Klassifikationsverfahren untersucht. Anschließend wird in Kapitel 4 das in dieser Arbeit genutzte Analyseverfahren beschrieben und begründet. Die zur Durchführung der Analyse konzipierte Software wird in Kapitel 5 beschrieben und evaluiert, um im Anschluss die Ergebnisse der Analyse in Kapitel 6 detailliert zu erörtern. Zum Abschluss werden in Kapitel 7 die gewonnenen Erkenntnisse zusammengefasst und Anpassungen der Messmethodik vorgeschlagen.

2. Grundlagen

Im Rahmen dieser Arbeit wird ein Datensatz untersucht, welcher den von 95 unbenutzten IPv4-Adressen im Netz des Universität Duisburg-Essen aufgezeichneten Internet-Datenverkehr enthält. Da diese Adressen nicht für die Bereitstellung von Diensten genutzt werden ist davon auszugehen, dass diese Aufzeichnungen keine sensiblen Benutzerinformationen enthält. Die Analyse der Daten sowie deren Ergebnisse müssen somit nicht unter besonderer Betrachtung von Datenschutzrichtlinien erfolgen.

2.1. Begrifflichkeiten

Die Begriffe *Client* und *Server* werden zumeist für die Bezeichnung zweier ungleicher Kommunikationspartner in Netzwerken genutzt. Der Client möchte dabei einen Dienst nutzen, welchen der Server zur Verfügung stellt. Es existieren jedoch auch andere Modelle wie Peer-to-Peer-Kommunikation, bei der beide Kommunikationspartner gleichgestellt sind. Im folgenden werden die Begriffe Client und Server auch für diese Modelle verwendet. Dabei ist der Client immer der Initiator der Kommunikation, auch wenn nach erfolgtem Verbindungsaufbau keine semantische Trennung mehr zwischen Client und Server im soeben beschriebenen Sinn vorliegt.

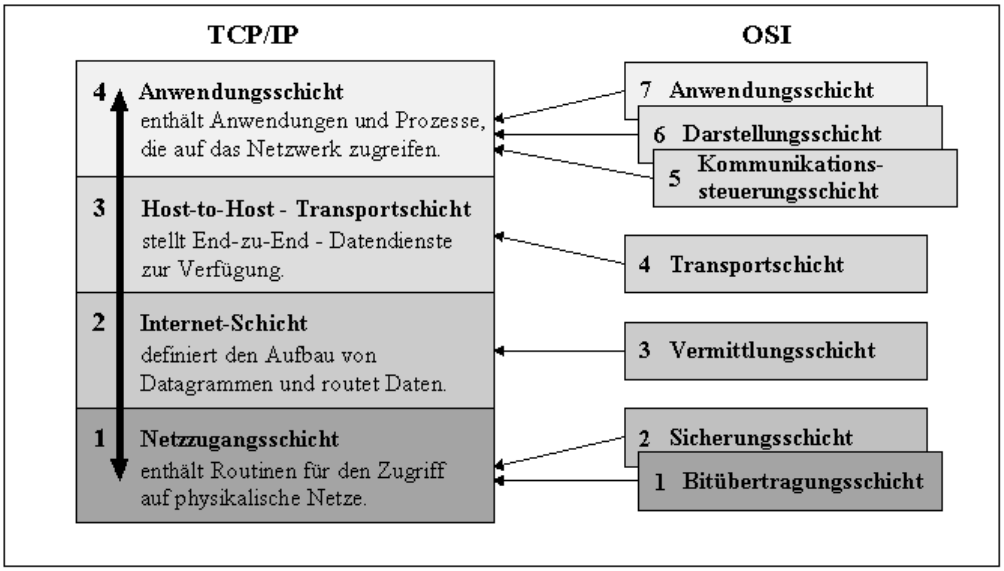
Durch die Verbreitung von Smartphones und des *Internet of Things* gibt es neben Servern in Rechenzentren und heimischen PCs viele weitere Geräte, welche über das Internet kommunizieren können. Alle diese netzwerkfähigen Geräte werden im folgenden unter dem Begriff *Computer* zusammengefasst.

2.2. Protokolle

Die Kommunikation in einem Computer-Netzwerk erfolgt über Protokolle. Diese definieren einheitliche Regeln zur Kommunikation und sorgen somit dafür, dass verschiedene Computer aus den übertragenen binären Daten die gleiche Information ableiten. Um die

Komplexität dieser Protokolle zu reduzieren, werden diese in mehreren Schichten organisiert. Jede Schicht implementiert dabei jeweils eine abgeschlossene Aufgabe innerhalb der Kommunikation und stellt eine Schnittstelle mit einer höheren Abstraktion für die nächste Schicht zur Verfügung.

Das ISO-OSI-Referenzmodell [Zim80] beschreibt als theoretisches Modell sieben Schichten. Die unterste Schicht, welche auch als *Layer 1* oder *Physical Layer* bezeichnet wird, definiert die physikalischen Parameter der Datenübertragung. Im Fall einer kabellosen Verbindung beinhaltet dies beispielsweise die verwendete Frequenz der Übertragungswelle. Für das Internet wird meist eine vereinfachte Darstellung verwendet, welche auf vier Schichten basiert. Das TCP/IP-Schichtenmodell fasst dabei die Schichten 1 und 2, sowie die Schichten 5, 6 und 7 des Referenzmodells zu je einer Schicht zusammen. Die Aufteilung und sowie die groben Aufgaben der Schichten sind in Abbildung 2.1 dargestellt.



Bildquelle: `http://www.ruhr-uni-bochum.de/~rothamcw/Lokale.Netze/tcpip.html`

Abbildung 2.1.: Zuordnung der Schichten des TCP/IP-Schichtenmodell zum ISO/OSI-Schichtenmodell

Jede Schicht überträgt neben den eigentlichen Daten, genannt *Payload*, zusätzliche Informationen, welche für die Steuerung der Kommunikation benötigt werden. Diese Information wird dabei direkt vor dem *Payload* übertragen und daher auch als Kopfdaten oder *Header* bezeichnet.

2.2.1. Ethernet

Das Ethernet-Protokoll IEEE 802.3 [Tan98] implementiert die Übertragungs- und Sicherungsschicht, also die unteren beiden Schichten des ISO-OSI-Referenzmodells, bzw. die Netzzugangsschicht des TCP/IP-Modells und ermöglicht somit die Übertragung von Daten zwischen zwei physikalisch verbundenen Computern. Der über die physikalische Leitung übertragene Ethernet-Frame besteht aus einer Preamble, einem Start Frame Delimiter (SFD) sowie dem Ethernet Frame.
Während die Preamble und der SFD zur Synchronisierung der Datenübertragung auf der physikalischen Leitung bestimmt sind und somit immer den gleichen Wert beinhalten, enthält der Ethernet-Frame die eigentlichen Informationen. Der Protokoll-Kopf des Frames besteht aus mindestens 14 Byte: einer sechs Byte langen "Destination MAC", welche den Empfänger eindeutig identifiziert, gefolgt von einer ebenfalls sechs Byte langen "Source MAC", welche den Sender identifiziert. Darauf folgen zwei Byte zur Angabe des Protokolls, welches in der nächsten Schicht verwendet wird. Sofern nicht abweichend angegeben, wird im folgenden davon ausgegangen, dass dieses den Wert `0x0800` enthält und somit auf das Internet Protokoll in der Version 4 (IPv4) verweist. Der optionale *VLAN Tag* ermöglicht den Betrieb mehrere virtueller Netze über eine einzelne Kabelverbindung, ist jedoch für diese Arbeit nicht von Bedeutung.

6	6	4	2	*lt* 1500	x
DST MAC	Source MAC	(VLAN Tag)	NP	Data	CRC

Abbildung 2.2.: Schematische Darstellung eines Ethernet Frames

2.2.2. Internet-Protokoll (IP)

Das Internet-Protokoll [RFC0791] implementiert die Netzwerkschicht des TCP/IP-Modells, bzw. die Vermittlungsschicht des ISO-OSI-Referenzmodells. Das Protokoll ermöglicht die Datenübertragung zwischen zwei Hosts, welche nicht direkt physikalisch verbunden sind. Das Protokoll wird derzeit in den Versionen 4 und 6 verwendet. Da die in dieser Arbeit betrachteten Daten ausschließlich auf IPv4 Datenverkehr beruhen, wird der Aufbau von IPv6 hier nicht näher betrachtet.

Zu den wichtigsten Feldern des IPv4-Headers gehören die Adressen von Absender und Empfänger, welche mit jeweils 32 Bit kodiert werden, sowie die Fragmentierungsinformationen.

Fragmentierung Ein IP-Paket wird fragmentiert wenn es größer ist als die maximal zulässige Größe des unterliegenden Frames der Netzzugangsschicht. Bei Verwendung des Ethernet Protokolls sind dies zumeist 1500 Byte. Das IP-Paket wird dann aufgeteilt und in mehreren Ethernet Frames verschickt. Um die Fragmente wieder zusammensetzen zu können, haben zusammengehörige Fragmente eine gemeinsame Kennung, welche im Identifikation-Feld kodiert wird. Das Tripel `(Quelladresse, Zieladresse, Identification)` ermöglicht dann die Wiederherstellung des IP-Pakets aus den Fragmenten. Die Position des Fragmentes innerhalb des Gesamtpakets wird durch den Fragmentation Offset angegeben. Ein zusätzliches Flag "More Fragments" markiert jeweils, dass noch weitere Fragmente zum zusammensetzen benötigt werden. Wird ein IP-Paket in mehreren Fragmenten empfangen, muss dieses zunächst zusammengesetzt werden, bevor es vollständig verarbeitet werden kann.

<table>
<tr><td>0-3</td><td>4-7</td><td>8-11</td><td>12-15</td><td>16-18</td><td>19-23</td><td>24-27</td><td>28-31</td></tr>
<tr><td>Vers.</td><td>IHL</td><td colspan="2">Type of Service</td><td colspan="4">Gesamtlänge</td></tr>
<tr><td colspan="4">Identifikation</td><td>Flags</td><td colspan="3">Fragment Offset</td></tr>
<tr><td colspan="2">TTL</td><td colspan="2">Protokoll</td><td colspan="4">Prüfsumme</td></tr>
<tr><td colspan="8">Quell IP-Adresse</td></tr>
<tr><td colspan="8">Ziel IP-Adresse</td></tr>
</table>

Abbildung 2.3.: Schematische Darstellung des IPv4 Headers

ICMP Die Kommunikation mittels IP ist grundsätzlich unidirektional. Zur Diagnose von Netzwerkproblemen und zur Signalisierung von Fehlern ist es jedoch notwendig dem Sender Informationen zurückzusenden. Für diesen Zweck existiert das Internet Control Message Protocol [RFC0792], welches zwar auf IP aufbaut, jedoch semantisch noch zur Vermittlungsschicht gezählt wird. Der ICMP- Header definiert die Art der Nachricht mittels zwei je ein Byte betragenden Feldern *Type* und *Code*. Je nach Nachrichtenart kann ein Payload mit weiteren Informationen übertragen werden. Ist etwa ein Paket nicht zustellbar, enthält der Payload des zur Signalisierung des Fehlers übertragenen ICMP-Pakets den Beginn des IP-Pakets, dessen Zustellung nicht möglich war.

Paketorientierung und Verbindungsorientierung

Eine Grundsätzliche Unterscheidung bei der Kommunikation in Netzwerken ist die zwischen paketorientierter, verbindungsloser Vermittlung (en: *packet switching*) und verbindungsorientierter Leitungsvermittlung (en: *circuit switching*). Bei paketorientierter Vermittlung kann grundsätzlich jedes versendete Paket auf einem anderen Weg durch das Netzwerk zum Empfänger zugestellt werden. Dadurch können die Pakete in einer

anderen Reihenfolge beim Empfänger ankommen, als sie vom Sender versendet wurden. Da es keine Zusicherung der Bandbreite gibt, ist ebenso nicht sichergestellt, dass die Pakete den Empfänger erreichen. Es ist möglich, dass diese bei einer Überlastung des Netzwerks verloren gehen.
Bei einer verbindungsorientierten Vermittlung hingegen wird ein fester Kanal zwischen zwei Kommunikationspartnern etabliert und für diesen eine Bandbreite zugesichert. Dadurch soll sichergestellt werden, dass alle Pakete in der richtigen Reihenfolge den Empfänger erreichen.

2.2.3. Transport-Protokolle

Transport-Protokolle stellen eine Kommunikationsverbindung zwischen zwei Prozessen her, welche auf verschiedenen Hosts ausgeführt werden können.

2.2.3.1. TCP

Das Transmission Control Protocol (TCP) [RFC0793b] ist ein Transport-Protokoll, welches aufbauend auf IP eine verbindungsorientierte Kommunikation simuliert. Aufgrund der zu Grunde liegenden Paketvermittlung kann TCP zwar nicht alle Eigenschaften einer Leitungsvermittlung – insbesondere die Zusicherung der Zustellung – garantieren, jedoch Paketverluste erkennen und versuchen, diese durch erneute Übertragung zu korrigieren. Zu den wichtigsten Aufgaben von TCP gehören der Auf- und Abbau einer Verbindung, die Sicherstellung einer vollständigen und fehlerfreien Datenübertragung, sowie Steuerung der Verbindungsgeschwindigkeit.
Der TCP-Header besteht aus mindestens 20 Byte, kann jedoch durch optionale Felder bis zu 60 Byte betragen. Dies wird durch das Feld `Offset` limitiert, welches den Beginn des ersten Datenblocks kennzeichnet.

<table>
<tr><td>0-3</td><td>4-7</td><td>8-15</td><td>16-31</td></tr>
<tr><td colspan="3">Quellport</td><td>Zielport</td></tr>
<tr><td colspan="4">Sequenznummer</td></tr>
<tr><td colspan="4">ACK-Nummer</td></tr>
<tr><td>Offset</td><td>Reserviert</td><td>Flags</td><td>Window</td></tr>
<tr><td colspan="3">Prüfsumme</td><td>Urgent pointer</td></tr>
<tr><td colspan="4">Optionen</td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

Abbildung 2.4.: Schematische Darstellung des TCP-Headers

Eine TCP-Verbindung lässt sich durch ein Quadrupel aus (Quell-IP, Quell-Port, Ziel-IP, Ziel-Port) identifizieren. Bevor eine Datenübertragung stattfinden kann, muss zunächst ein Verbindungsaufbau durchgeführt werden, welcher als *three-way-handshake* bezeichnet wird. Dabei sendet der Client ein Paket an den Server, welches das Synchronize Flag (`SYN`) gesetzt hat, initiiert den Verbindungsaufbau in die Senderichtung vom Client zum Server (C2S). Das Paket enthält eine Sequenznummer `x`. Als Antwort darauf sendet der Server ein Paket zum Client, welches zwei Merkmale hat. Das erste Merkmal ist, dass das Acknowledgement Flag (`ACK`) gesetzt ist und die Acknowledgement-Nummer den Wert `x+1` enthält. Dadurch bestätigt der Server den Verbindungsaufbau in Richtung C2S. Das zweite Merkmal ist das ebenfalls gesetzt `SYN` Flag, sowie die Sequenz-Nummer `y`, wodurch die Verbindung in Gegenrichtung Server zu Client (S2C) aufgebaut wird. Im dritten Schritt des Verbindungsaufbaus, bestätigt der Client den Verbindungsaufbau in S2C durch ein Paket mit gesetztem `ACK` Flag und der Acknowledgement-Nummer `y+1`, sowie der Sequenznummer `x+1`.

Der zur Bestätigung verwendete Mechanismus wird grundsätzlich bei allen Paketen angewendet. Eine Empfangsbestätigung erfolgt dadurch, dass ein Paket mit gesetztem ACK-Flag versendet wird, welches als Acknowledgement-Nummer den Wert der bestätigten Sequenznummer plus eins einhält. Eine Bestätigung kann dabei auch für mehrere Pakete zusammen erfolgen. Die Acknowledgement-Nummer bezieht sich dabei jeweils immer auf das letzte bestätigte Paket und beinhaltet alle vorausgegangenen.

2.2.3.2. UDP

Im Gegensatz zu TCP ist das *User Datagram Protocol (UDP)* [RFC0793a] ein verbindungsloses Transportprotokoll, welches keine Zusicherungen für die fehlerfreie Zustellung von Paketen bietet. Der *Header* eines UDP-Datagramms beinhaltet neben Quell- und Zielport zur Identifikation der Anwendung lediglich die Länge des *Datagrams* sowie eine Prüfsumme zur Erkennung der fehlerhaften Übertragung des einzelnen *Datagrams*. Fehlerhafte Übertragungen werden durch UDP jedoch nicht erneut übertragen. Ebenso stellt UDP nicht sicher, dass die übertragenen Datagramme in der richtigen Reihenfolge beim Empfänger ankommen.

Da UDP im Gegensatz zu TCP keinen Verbindungsaufbau benötigt, wird es in Anwendungen eingesetzt, welche nur kurzzeitige Kommunikation im Frage-Antwort-Schema benötigen. Ebenso kommt es für die Übertragung von Audio- und Videodaten zum Einsatz, da hier eine geringe Latenz wichtiger ist als die eine vollständig fehlerfreie Übertragung.

2.2.4. Anwendungsprotokolle

2.2.4.1. Hypertext Transfer Protocol

Das am häufigsten eingesetzte Anwendungsprotokoll im Internet ist das Hypertext Transfer Protocol (HTTP), welches 1996 von Tim Berners-Lee [RFC1945] in der Version 1.0 beschrieben und entwickelt wurde. Die überarbeitete Version 1.1 [RFC2616] wird seit 1999 verwendet. Seit 2015 ist die Nachfolgerversion HTTP 2.0 spezifiziert, welche jedoch im hier betrachteten Messzeitraum kaum verbreitet ist [Var+16]. HTTP ist ein Textprotokoll und ermöglicht Dokumente über eine feste Adresse abzurufen oder zu modifizieren. Das Protokoll ist grundsätzlich zustandslos. Eine Anfrage des Clienten wird vom Server unabhängig von vorherigen Anfragen beantwortet. Die Anfrage besteht im einfachen Fall aus einer Zugriffsmethode, einer Adresse und der HTTP-Version. Die Zugriffsmethode beschreibt, ob das Dokument abgerufen (`GET`), modifiziert (`PUT, POST, PATCH`) oder gelöscht (`DELETE`) werden soll. Die Adresse spezifiziert das Dokument, auf welcher die Operation stattfindet. Zusätzlich können diverse zusätzliche Kopfdaten der Form `Name: Wert` übertragen werden, welche Details, wie etwa das akzeptierte Datenformat festlegen. Bei modifizierenden Anfragen existiert ebenfalls ein Anfragen-Körper (*Request Body*), welcher die Modifikationen enthält.
Die Antwort des Servers besteht aus deiner Statuszeile, welche den Erfolg der Anfrage oder verschiedene Spezial- und Fehlerfälle definiert, etwa wenn das Dokument nicht existiert oder sich dessen Adresse geändert hat. Auf die Statuszeile folgen diverse Kopfdaten, sowie als *Response Body* das Dokument.

Der Standardport für HTTP ist 80, jedoch sind auch alternative Ports wie 8000 oder 8080 geläufig. Wird HTTP aufbauend auf dem im folgenden beschriebenen TLS eingesetzt wird diese Protokollkombination auch als HTTPS bezeichnet. Für HTTPS ist der TCP-Port 443 verwendet.

2.2.4.2. Transport Layer Security

Durch das Protokoll *Transport Layer Security (TLS)*, welches derzeit in der Version 1.2 verwendet wird [RFC5246], wird zwischen dem Transport- und dem Anwendungsprotokoll eine Schicht etabliert, welche Integrität und Vertraulichkeit der übertragenen Daten zusichert [RFC2246]. Dies bedeutet, dass über TLS übertragene Daten von Dritten weder gelesen, noch unbemerkt geändert werden können. Durch den zusätzlichen Einsatz einer *Public-Key-Infrastructure (PKI)* soll weiterhin die Authentizität der Daten sichergestellt werden. Der Vorgänger von TLS, *Secure Socket Layer (SSL)* gilt aufgrund verschiedener Sicherheitslücken als unsicher und sollte daher nicht mehr verwendet werden.

TLS setzt auf einem verlässlichen Transport-Protokoll, wie etwa TCP, auf und führt darauf aufbauend einen eigenen Verbindungsaufbau (*handshake*) durch, bei welchem die Details der gesicherten Verbindung, wie etwa das verwendete Verschlüsselungsverfahren und die verwendeten Schlüssel ausgetauscht werden. Initiiert wird dieser Verbindungsaufbau durch das *ClientHello* Paket des Clients.

Die gängigsten Anwendungsprotokolle, welche TLS nutzen, sind HTTPS (Port 443), sowie für Mailversand- und Empfang eingesetzte Protokolle wie SMTPS (Port 465) oder IMAPS (Port 993). Für Mail-Protokolle existiert weiterhin ein *STARTTLS* genanntes Verfahren, bei welchem zunächst eine ungeschützte TCP-Verbindung zwischen Client und Server aufgebaut wird, welche im Nachhinein einen TLS-Verbindungsaufbau durchführt.

2.2.4.3. DNS

Das *Domain Name System (DNS)* [RFC1035] ist ein hierarchisch organisiertes Verzeichnissystem, dessen primäre Aufgabe die Auflösung von Namen in IP-Adressen ist. Die Kommunikation zwischen einem Client und einem einzelnen DNS-Server, auch *Nameserver* genannt, folgt dabei einem Fragen-Antwort-Schema. Der Client sendet eine Anfrage nach der gewünschten Information auf UDP-Port 53 an den Server, welche dieser mit einer einzelnen Antwort erwidert. Für die Beantwortung großer Anfragen kann ebenfalls der TCP-Port 53 genutzt werden. Gegen DNS sind verschiedene Angriffszenarien bekannt. Dazu gehören Denial-of-Service Angriffe gegen Nameserver, bei welchem dieser mit vielen Anfragen überlastet werden soll.

2.2.4.4. FTP

Über das *File Transfer Protocol (FTP)* ist die Übertragung von Dateien von und zu anderen Computern möglich. Das Protokoll nutzt TCP als Transportschicht, wobei zwei Ports verwendet werden. Der initiale Port 21 wird zur Übertragung von Steuerdaten verwendet. Die Übertragung des Dateiinhaltes erfolgt über den Port 20. Da das Protokoll keine Limitierung der Authentifizierungsversuche vorsieht [RFC2577] ist es anfällig für "*password guessing*", also den Versuch, systematisch viele mögliche Passwörter auszuprobieren, um sich unbefugt Zugriff zum System zu verschaffen.

2.2.4.5. Telnet und SSH

Die Protokolle Telnet [RFC0854] und *Secure Shell (SSH)* ermöglichen den Zugriff auf die Konsole eines entfernten Computers über ein Netzwerk um diesen zu steuern. Während das auf TCP-Port 23 laufende Telnet sowohl Login-Informationen als auch übertragende Befehle und Rückmeldungen unverschlüsselt überträgt, etabliert SSH eine Sicherheitsschicht, welche sowohl Vertraulichkeit, als auch eine Authentifizierung in beide Richtungen sicherstellt. Bevor der Client ein Passwort sendet, muss sich zunächst der Server gegenüber dem Client authentifizieren. SSH nutzt als Standard den TCP-Port 22 und ermöglicht neben der Steuerung mittels Shell auch die Übertragung von Dateien (SFTP), wodurch es eine sichere Alternative zum zuvor beschriebenen FTP darstellt.

2.2.4.6. BitTorrent

Das Protokoll *BitTorrent* ist ein *Peer-to-Peer (P2P)* Protokoll zum verbreiten von Dateien. Die Dateiübertragung erfolgt dabei nicht wie in FTP zwischen einem Client und einem Server, sondern zwischen allen Teilnehmern eines sogenannten *Swarms*. Zur Koordination des Swarms bietet BitTorrent mehrere Möglichkeiten, darunter die Nutzung eines zentralen Koordinators (*Tracker*) oder die dezentrale Koordination durch die Verwendung einer *Distrubuted Hash Table (DHT)*.

Für die Kommunikation zwischen den einzelnen Peers zum Dateiaustausch wird als Transportprotokoll TCP genutzt. Der Port ist dabei nicht fest, sondern wird dynamisch im Bereich 6881-6889 ausgewählt [BEP0003], da dieser Portbereich von anderen Prozessen für ausgehende Verbindungen genutzt werden kann. Der Standard schreibt diesen Portbereich jedoch nicht fest vor, sondern definiert dieses Vorgehen lediglich als übliches Vorgehen, sodass grundsätzlich auch andere Ports genutzt werden können.

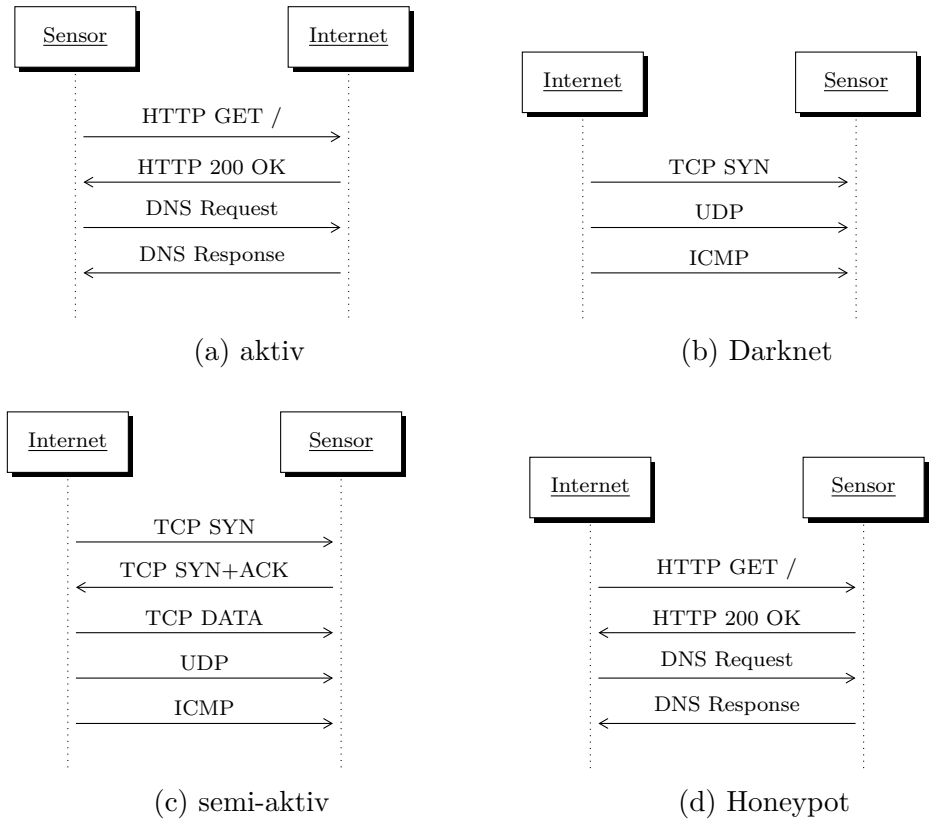
Als Transport-Protokoll für andere Bestandteile von BitTorrent, wie die Verwendung von DHTs [BEP0005] und dem PEX-Verfahren [BEP0011] zum gegenseitigen Austausch von Peers kommt UDP zum Einsatz.

2.3. Netzwerk-Messungen

Die Analyse von Internet-Datenverkehr erfordert die Möglichkeit des Mitschnitts der versendeten Datenpakete. Eine Möglichkeit diesen Mitschnitt zu erhalten ist die Aufzeichnung an einem Knotenpunkt, welcher den Datenverkehr für andere Netzwerkteilnehmer weiterleitet. Dies ist jedoch aus mehreren Gründen oft ungeeignet. Zum Einen

verarbeiten diese Knotenpunkte sehr große Datenmengen. Dadurch wird eine Analyse der Daten Zeit- und Ressourcen-intensiv und somit unpraktikabel. Zum Anderen enthält der Datenverkehr sensible Daten, sodass ein Mitschnitt aus Gründen des Datenschutzes nicht möglich ist.

Eine alternative Messung kann dadurch erfolgen, dass der Messsensor selbst ein Computer im Netzwerk ist, dessen ein- und ausgehender Datenverkehr analysiert werden sollen. Dies reduziert im Vergleich zum vorherigen Verfahren die zu analysierende Datenmenge. Die Auswirkung auf datenschutzrechtliche Limitierungen ist jedoch abhängig von der genauen Gestaltung des Messverfahrens. Grundsätzlich unterschieden wird hier zwischen der aktiven und der passiven Messung, wobei verschiedene Zwischenabstufungen möglich sind. Dazu gehören aktives Antworten sowie *Honeypots*.



(a) aktiv (b) Darknet

(c) semi-aktiv (d) Honeypot

Abbildung 2.5.: Darstellung verschiedener Messmethoden

2.3.1. Aktive Messungen

Bei einer aktiven Messung (Abbildung 2.5a) ist ein Sensor der Client einer Netzwerkverbindung. Er sendet Pakete zu anderen Netzwerkteilnehmern um deren Antwort analysieren zu können. Anhand der Antwort, bzw. dem Ausbleiben selbiger, kann der Sensor die

Verbreitung bestimmter Protokolle oder Protokollversionen analysieren. Um die Aussagekraft der Analyse zu erhöhen werden hierzu Anfragen an möglichst viele andere Netzwerkteilnehmer gesendet. Dieses als *Scanning* bezeichnete Vorgehen wird jedoch nicht nur für wissenschaftliche Messungen, sondern auch für die strukturierte Suche nach, bzw. auch das Ausnutzen von Angriffspunkten auf anderen Computern verwendet.

2.3.2. Passive Messungen

Bei einer passiven Messung versendet der Sensor keine Pakete im Netzwerk sondern empfängt nur solche, welche von anderen Teilnehmern zu ihm geschickt werden. Bei einem Sonderfall dieser Art der Messung werden Adressen ausgewählt, welche weder für die Bereitstellung von Diensten noch für ausgehende Verbindungen von Clients verwendet werden. Man spricht dann von einem *Darknet* (Abbildung 2.5b). Da diese Adressen somit nicht für gewöhnlichen Datenverkehr benutzt werden, sollte der Sensor theoretisch gar keine Daten empfangen. Es gibt jedoch verschiedene Effekte, die dazu führen, dass der passive Sensor dennoch Daten aufzeichnet. Einer dieser Effekte sind die zuvor erwähnten aktiven Messungen. Wenn diese den gesamten möglichen Adressraum untersuchen, schließt dies auch die Adressen eines passiven Sensors mit ein. Neben *Scanning* können auch Fehlkonfigurationen oder Artefakte von *Denial-of-Service (DoS)* Angriffen von einem *Darknet* aufgezeichnet werden. *DoS* Angriffe werden in Abschnitt 2.4 genauer beschrieben. Die Menge dieser Effekte, welche von einem reinen *Darknet* empfangen werden können, werden als Hintergrundstrahlung oder *Background radiation* bezeichnet.

Diese Auswahl der Adressen hat gegenüber der Messung an einem Netzwerkknoten den Vorteil, dass eine implizite Filterung des Datenverkehrs vorgenommen wird. Der Knotenpunkt verarbeitet alle Daten, die zwischen aktiven Netzen ausgetauscht werden. Diese bestehen primär aus regulärem Datenverkehr. Bei einem *Darknet* tritt hingegen kein regulären Datenverkehr auf. Die dennoch aufgezeichneten Daten sind daher größtenteils Hintergrundstrahlung, welche von besonderem Interesse ist. Für *Darknets* sind in der Literatur verschiedene Begriffe zu finden, welche im Kern die gleiche Technik beschreiben. Dazu gehören "*Network Telescope*", "*Internet motion sensor*" und "*Black hole*" [HA05].

Passive Messungen unterliegen jedoch auch Einschränkungen. Zum einen können diese nur Pakete messen, welche an den von ihnen abgedeckten Adressraum gesendet werden. Passive Messungen werden daher oft mit sehr großen Adressräumen verwendet um die Menge des empfangenen Datenverkehrs zu erhöhen. Eine weitere Einschränkung wird durch die verwendeten Protokolle verursacht. Manche Informationen werden in diesen erst durch eine wechselseitige Kommunikation übertragen. Um diese Informationen zu

erhalten, muss der Sensor auf empfangene Pakete antworten. Das Verfahren wird als *active responding*, bzw. aktives Antworten bezeichnet und kann in verschiedenen Schichten des Protokollstacks etabliert werden.

Transportschicht Aktives Antworten in der Transportschicht wird meist für das Transport-Protokoll TCP eingesetzt. Dieses ist verbindungsorientiert und erfordert zunächst einen Verbindungsaufbau, bei welchem Client und Server sich gegenseitig Pakete zusenden, bevor der Austausch der eigentlichen Nutzdaten stattfindet. Da das Senden passender Antworten zu eingehenden Paketen jeweils direkt aus dem empfangenen Paket hergeleitet werden kann und nicht der Zustand der kompletten TCP-Verbindung gespeichert werden muss, lässt sich *active responding* für TCP einfach und effizient implementieren. Es wird grundsätzlich mit einer Bestätigung (*acknowledgement*) des empfangenen Pakets geantwortet. Auf eine Verbindungsanfrage (`SYN`) folgt ein `SYN ACK`, auf Datenpaket ein `ACK` um den Empfang zu quittieren und auf einen Verbindungsabbau (`FIN`) wird ein `FIN ACK` geantwortet.

Ohne aktives Antworten auf der Transportschicht empfängt der Sensor bei eingehenden TCP-Verbindungen jeweils nur das initiale `SYN` Paket, aus dem sich etwa der Zielport auslesen lässt, welcher jedoch nur ein Hinweis und kein hinreichendes Kriterium für die Bestimmung des genutzten Anwendungsprotokolls ist. Durch das Versenden von Antworten kann der Client den eigentlichen Payload der Verbindung senden, sodass sich aus diesem Informationen über das genutzte Anwendungsprotokoll auslesen lassen können. Auch der Versuch Schwachstellen in Implementierungen des Anwendungsprotokolls auszunutzen kann in bestimmten Fällen aus dem Payload ausgelesen werden.

Anwendungsschicht Manche Angriffe lassen sich jedoch erst erkennen, wenn der Sensor in der Anwendungsschicht ein bestimmtes Antwortverhalten zeigt. Dieses Antwortverhalten muss für jedes untersuchte Anwendungsprotokoll definiert werden. Aufgrund der Vielzahl möglicher Anwendungsprotokolle kann dieses Verfahren nur gezielt für eine Auswahl von Protokollen realistisch durchgeführt werden. Für manche Anwendungsprotokolle lässt sich - wie schon in der Transportschicht - eine passende Antwort aus der empfangenen Anfrage herleiten, sodass kein Zustand der Verbindung gespeichert werden muss.

Honeypot Wird das Anwendungsprotokoll vollständig simuliert, bezeichnet man den Sensor auch als *Honeypot* (Abbildung 2.5d), bzw. *Honeynet*, falls es sich um ein Sensorennetzwerk handelt. Mittels eines Honeypots lassen sich Angriffe auf bestimmte Implementierungen eines Protokolls untersuchen. Diese Implementierung wird dazu in einer

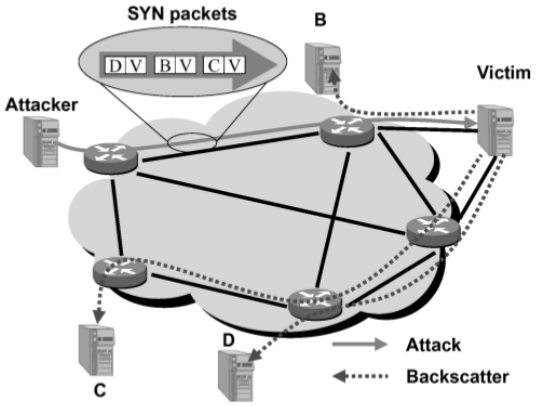
gesicherten Umgebung installiert und kann dann über das Netzwerk angegriffen werden. Die Absicherung erfolgt, damit ein erfolgreicher Angreifer nicht vom kompromittierten Honeypot in andere Systeme vordringen kann.

Semi-aktive Messungen Während passive Messungen den gesamten IP Datenverkehr erfassen, schränken aktive Messungen, sowie aktives Antworten in der Anwendungsschicht die Messung auf eine festgelegte Auswahl an Protokollen ein. Aktives Antworten in der Transportschicht muss zwar theoretisch auch für alle Protokolle durchgeführt werden, ist in der Praxis jedoch nicht notwendig. Über 99% des auf IP basierenden Traffics werden durch die drei Protokolle TCP, UDP und ICMP abgedeckt [Wus+10]. Von diesen Protokollen benötigt nur TCP einen aktiven Responder, damit der Sensor den Payload empfangen kann.

Bei aktivem Antworten in der Transportschicht verhält sich der Sensor passiv für UDP und ICMP-Datenverkehr. Aus Sicht der Anwendungsschicht zeigt der Sensor ebenfalls passives Messverhalten. Für TCP-Datenverkehr zeigt der Sensor initiativ kein aktives Verhalten, wird jedoch aktiv, sobald eingehender Datenverkehr dies erforderlich macht. Das heißt, er antwortet auf eingehende Verbindungsversuche und bestätigt den Empfang eingehender Datenpakete. Dieses Verhalten wird im Rahmen dieser Arbeit als *semi-aktiv* (Abbildung 2.5c) bezeichnet.

2.4. Denial-of-Service

Bei einem *Denial-of-Service (DoS)* [Moo+06] Angriff versucht ein Angreifer durch das Versenden großen Anfragemengen einen Server zu überlasten, sodass dieser keine Anfragen mehr beantworten kann. Eine mögliche Gegenmaßnahme für diese Art von Angriffen ist das Sperren der anfragenden IP-Adressen, sodass diese keine weiteren Daten an den überlasteten Dienst, bzw das überlastete Netzwerk versenden können. Diese Gegenmaßnahme ist jedoch wenig effektiv, falls der Angreifer die Absender-Adresse der von ihm verschickten IP-Pakete fälscht. Diese als *IP-Spoofing* bekannte Technik ermöglicht es dem Angreifer Sperren seiner eigenen IP-Adresse bei einem *DoS*-Angriff zu umgehen. Da der angegriffene Dienst weiterhin versucht, die eingehenden Anfragen zu beantworten, werden dessen Antworten jedoch an die vom Angreifer gefälschte Adresse gesendet. Diese Antworten werden als *Backscatter* bezeichnet. Wie in Abbildung 2.6 gezeigt, versendet der Angreifer Pakete mit den Quelladressen von B, C und D an das Ziel V. Die Antworten des Servers werden anschließend zu diesen bisher an der Kommunikation unbeteiligten Netzwerkteilnehmern gesendet.



Bildquelle: [Moo+06]

Abbildung 2.6.: Darstellung eines Denial-of-Service Angriffs mit Backscatter

2.5. Packet Capture (PCAP)

Das PCAP-Dateiformat ermöglicht das Speichern von Netzwerkmitschnitten als Binärdaten. Eine PCAP-Datei besteht aus einem globalen Header mit einer Größe von 24 Byte und einer Liste von Null oder mehr *Records* [pcap]. Jeder Record besteht wiederum aus einem vier Byte langem Paket-Header sowie den eigentlichen Paketdaten der mitgeschnittenen Netzwerkschnittstelle, deren Aufbau in Abschnitt 2.2 beschrieben wurde.

Globaler Header Der globale Header einer PCAP-Datei besteht aus sieben Werten, welche wie in Abbildung 2.7 gezeigt angeordnet sind. Die Erkennung des Dateiformats wird durch eine Versionsnummer, sowie eine `magic_number` ermöglicht, welche mit einem festen Wert belegt wird. Durch die `magic_number` wird zusätzlich die Auflösung der Zeitstempel – `0xa1b2c3d4` für Mikrosekunden- und `0xa1b23c4d` für Nanosekunden-Genauigkeit – sowie die Byte-Order der Datei festgelegt.
Zur Korrektur der Zeitstempel in der Datei wird im globalen Header weiterhin die Verschiebung der Zeitzone zur UTC in Sekunden (`thiszone`) und die Genauigkeit der Zeitstempel (`sigfigs`) angegeben. Der Wert sigfigs wird jedoch zumeist nicht genutzt und ist in vielen Implementierungen mit der `0` belegt. Mittels des Feldes `network` wird der Typ des LinkLayer-Headers angegeben, dessen Mitschnitt in der Datei enthalten ist. Das Dateiformat unterstützt weiterhin partielle Mitschnitte des Datenverkehrs. Durch `snaplen` wird die maximale Größe eines mitgeschnittenen Pakets definiert.

Paket Header Der Paket Header jedes *Records* enthält den Zeitpunkt und die Größe des mitgeschnittenen Pakets. Der Zeitpunkt wird in zwei Werten zu je vier Byte kodiert,

0-1	2-3	4-5	6-7
magic_number		version_major	version_minor
thiszone		sigfigs	
snaplen		network	

Abbildung 2.7.: Schematische Darstellung des global PCAP Datei-Headers

einem Zeitstempel in Sekunden `ts_sec`, welcher der UNIX-Time entspricht, sowie den zusätzlich in dieser Sekunde vergangenen Mikro-, bzw. Nanosekunden.
Bei der Größe des Paketes wird zwischen der tatsächlichen Größe des Pakets `orig_len` und der Größe des gespeicherten Mitschnitts `incl_len` unterschieden. Falls `orig_len` größer als die im globalen Header festgelegte maximale Mitschnittsgröße ist, werden nur die ersten `snaplen` Bytes des Pakets in der Datei gespeichert und es gilt `incl_len = snaplen`. Andernfalls sind `orig_len` und `incl_len` identisch.

0-3	4-7
`ts_sec`	`ts_usec`
`incl_len`	`orig_len`

Abbildung 2.8.: Schematische Darstellung des global PCAP Datei-Headers

3. Verwandte Arbeiten

Es existieren bereits einige Forschungsarbeiten zur Analyse und Klassifikation von Netzwerk-Messungen. Diese unterscheiden sich im Hinblick auf eingesetzte Messmethodik und Klassifizierungsverfahren.

3.1. Messmethodik

Eine der ersten Veröffentlichungen zur Analyse von Internet-Hintergrundstrahlung stammt von Moore et. al. aus dem Jahr 2004 [Moo+04]. Die Forscher führen dabei den Begriff des *Network Telescope* als Analogie zu einem astronomischen Teleskop ein. Das Netzwerk-Teleskop beobachtet einen Bereich des Internets und empfängt die dort eintreffenden Pakete. Als entscheidenden Faktor für die Auflösung der Messung wird die Größe des Sensors genannt. Die Größe ist bei Netzwerk-Teleskopen durch den abgedeckten Adressraum, also der Anzahl von IP-Adressen gegeben, welche das Messnetzwerk umfasst.
Die Arbeit benennt bereits Probleme dieses Analysemodells und unterbreitet mögliche Lösungsansätze. Als Hauptproblem wird gesehen, dass Ereignisse aufgrund verschiedener Ereignisse nicht vom Teleskop erfasst werden können. Die Gründe hierfür können unter anderem Zufall, bzw. ein Bias sein, wodurch die Effekte nicht in dem beobachteten Netzwerkbereich auftreten, aber auch überlastete, bzw. ausgefallene Netzwerke. Die vorgeschlagenen Lösungen hierfür sind *distributed network telesopes*, also auf mehrere Adressbereiche verteilte Teleskope, welche sich über den verfügbaren IPv4 Adressraum verteilen und *anycast network telescopes*, bei welchen ein Adressbereich von mehreren Teleskopen beobachtet wird. Als weiteres Problem wird genannt, dass die rein passive Messung nur beschränkte Informationen zu manchen Ereignissen liefert. Die genannte Lösung hierfür ist das *honeyfarm telescope*, welches aktiv auf eingehende Pakete antwortet um weitergehende Informationen zu sammeln.

Im weiteren werden Analysen genannt, welche jeweils unterschiedlich ausgeprägte *active responder* verwenden. Diese lassen sich grob in drei Kategorien unterteilen. Die erste

Kategorie sind rein passive Messungen, welche keinen *active responder* einsetzen. Daraufhin werden Messungen betrachtet, welche einen *active responder* in der Transportschicht einsetzten. Die dritte Kategorie umfasst Messungen, welche für ausgewählte Protokolle Antworten auf Applikationsebene versenden. Neben dem Begriff des *network telescope* werden für diese Art der Messung auch die Bezeichnungen *blackhole monitor*, *network sink*, *network motion sensor* oder *darknet* in der Literatur verwendet [HA05].

3.1.1. Passive Messungen

Die größte Analyse von Internet-Hintergrundstrahlung im IPv4 Netz mit passiver Messung führten Wustrow et. al. [Wus+10] 2010 durch. Ein /8 IPv4 Netzwerk, welches rund 16 Millionen Adressen beinhaltet, wurde über einen Zeitraum von etwa fünf Jahren als *darknet* betrieben. Zusätzlich wurden über mehrere kurze Messphasen von jeweils einer Woche vier verschiedene /8 IPv4 Netzwerke genutzt. In der Auswertung der Langzeitmessung haben die Forscher einen Rückgang von `TCP SYN+ACK` und einen Anstieg von `TCP SYN` Paketen festgestellt, was nach deren Klassifikation einem Anstieg von Scanning Aktivitäten und einem Rückgang von Backscatter Artefakten im gemessenen Datenverkehr entspricht.
Aus den Messdaten der Kurzzeitmessungen konnten die Forscher starke Abweichungen innerhalb bestimmter Subnetze erkennen. Diese als *Internet Address Pollution* bezeichneten Effekte werden in Abschnitt 3.3 beschrieben.

Auf Basis eines Datensatzes, der von einem „*lightly utilized /8*“ großen IPv4 Netzwerk aufgezeichnet wurde, untersuchten Moore et. al. [Moo+06] die Verbreitung von Denial-of-Service (DoS) Angriffen anhand von Backscatter Artefakten, welche aufgrund gefälschter Quelladressen des Angreifers an das Messnetzwerk gesendet werden. Über einen Zeitraum von 3 Jahren wurden dabei Messungen verschiedener Messperioden von fünf bis 18 Tagen mit einem Gesamtvolumen von etwa 210 Tagen ausgewertet. Mittels der eingesetzten Messmethode wurde in dieser Studie erstmals der globale Umfang von DoS-Angriffen quantifiziert.

Durumeric et. al. [DBH14] untersuchten 2014 mittels eines rund 5,5 Millionen IPv4 Adressen umfassenden Darknets, welches über einen Zeitraum von 16 Monaten eingehenden Datenverkehr aufgezeichnet hat, den Umfang und die Charakteristiken von Scanning im Internet. Dabei wurde die Definition eines Scans präziser gefasst als bei Wustrow. Während dort das Auftreten eines `TCP SYN`-Pakets als Scanning-Aktivität gewertet wurde, setzt Durumeric für die Einordnung als Scan voraus, dass von der Absenderadresse mindestens 100 Zieladressen des Messnetzwerks auf dem gleichen Port mit einer Rate von mindestens zehn Paketen pro Sekunde angesprochen werden. Die Auf dieser Basis

erkannten Scans wurden unter anderem bezüglich der am meisten gescannten Protokolle, sowie des Herkunftslandes des Scans hin untersucht.
Weiterhin wurde eine rudimentäre Klassifikation der für den Scan genutzten Software vorgenommen, welche auf dem *identification* Feld des IPv4 Headers beruht. Für Masscan [Gra] und ZMap [DWH13] lässt sich dieser Wert zur Identifikation des Scanners nutzen. ZMap verwendet den festen Wert `54321`. In Masscan wird der Wert durch die Vorschrift $ip\_id = dst\_addr \oplus dst\_port \oplus tcp\_seqnum$ gebildet.

Mit der Erkennung von Malware Ausbrüchen mittels *blackhole traffic* haben sich Soltani et. al. [SKR08] beschäftigt. Ein *darknet*, welches mehrere /24 IPv4 Netzwerke umfasst wurde über einen Zeitraum von 345 Tagen durch einen Detektor ergänzt, welcher die eingehenden Daten aufgrund statistischer Modelle live analysiert. Anhand statistischer Abweichungen wurden im Messzeitraum zwölf Ausbrüche von Computer-Würmern erkannt. Der Erkennungszeitpunkt mittels des vorgeschlagenen Verfahrens ist in den meisten Fällen wenige Tage vor der Auffindung im *Symantec Internet Security Threat Report*.

Die meisten Untersuchungen von *background radiation* beschränken sich auf die Untersuchung von IPv4 Datenverkehr. Aufgrund des stark beschränkten Adressraums von rund vier Milliarden IPv4 Adressen sind selbst große *darknets* auf wenige Prozent des Adressraums beschränkt. Dies ist darin begründet, dass ein Großteil des verfügbaren IPv4-Adressraums aktiv genutzt wird und somit nicht für Forschnungsmessungen zur Verfügung steht. Der IPv6 Adressraum unterliegt aufgrund der Gesamtgröße von 2^{128} möglichen Adressen nicht so starken Einschränkungen. Bei einer Untersuchung von Czyz et. al. im Jahr 2013 [Czy+13] wurde über einen Zeitraum von drei Monaten ein Darknet genutzt, welches aus mehreren /12 IPv6 Netzwerken besteht. Nach Angabe der Autoren waren dies zum Zeitpunkt der Messung über 80% des zugewiesenen IPv6 Adressbereichs. Die Analyse der Daten ergab signifikante Unterschiede der Hintergrundstrahlung zwischen IPv4 und IPv6 in Volumen und Art. Die Paketrate im IPv6 Netz war trotz dessen deutlich größeren Umfangs um den Faktor 500 geringer als beim zum Vergleich genutzten /8 IPv4 Netz. Während in IPv4 *network telescopes* über 80% des Traffics das Transport Protokoll TCP nutzen wurden bei IPv6 primär ICMP-Pakete gemessen. Dieses deutet darauf hin, dass IPv6 bisher noch nicht im gleichen Ausmaß wie das Vorgängerprotokoll von bösartigem Datenverkehr betroffen ist und sich die Hintergrundstrahlung zum Großteil aus „*probing, diagnostic, and management traffic*“ zusammensetzt.

3.1.2. Semi-aktive Messungen

Die größte semi-aktive Messung führten Bailey et. al. [Bai+05] durch. Dazu wurde nicht nur ein einzelnes Netzwerk verwendet, sondern wie zuvor von Moore vorgeschlagen, ein

distrubuted network telescope konzipiert, welches Bailey unter dem Namen „*internet motion sensor*“ vorstellt. Das verteilte System umfasst 28 Netzwerke mit Größen zwischen /25 und /8, welche insgesamt rund 17 Millionen IPv4 Adressen beinhalten. Der eingesetzte „*lightweight active responder*“ beantwortet eingehende TCP-Verbindungsanfragen (`TCP SYN`) mit einem `TCP SYN-ACK`, wodurch ein Verbindungsaufbau erfolgreich ist und der Absender den eigentlichen Payload versendet. Der Erhalt des Payloads wird jedoch nicht mittels eines `TCP ACK` beantwortet, wodurch sich die Implementierung an dieser Stelle nicht protokollkonform verhält.
Die Vorteile des *active responders* werden in der Möglichkeit gesehen, den Payload von TCP-Anfragen messen und dadurch die Ursache genau erkennen zu können. Dies gilt insbesondere für Angriffe auf bestimmte Schwachstellen in Applikationsprotokollen, welche allein durch den Port im `SYN`-Paket nicht eindeutig identifiziert werden können. Das Paper benennt jedoch auch einen Nachteil dieser Messmethode. Dadurch, dass der *active responder* alle eingehenden Anfragen unabhängig vom Port beantwortet, lässt sich der Sensor leicht erkennen und kann so prinzipiell von den Erzeugern schadhaften Datenverkehrs gemieden werden, um weniger auswertbare Daten zu hinterlassen.

3.1.3. Aktives Antworten auf Anwendungsebene

Einen weitergehenden Ansatz beim Beantworten von Anfragen, welche vom vom Netzwerk Teleskop empfangen werden, haben Yegneswaran et. al. [YBP04] umgesetzt. Neben der Beantwortung von TCP-Verbindungsanfragen wurden ebenfalls applikationsspezifische Antworten für HTTP, NetBIOS/Samba, SMTP, IRC und weitere Anwendungsprotokolle versendet. Um eine hohe Skalierbarkeit zu gewährleisten wurden die *responder* zustandslos konzipiert. Die Autoren postulieren dazu die Annahme, dass eine geeignete Antwort auch ohne das Vorwissen der bisherigen Konversation und nur aufgrund der einzelnen Anfrage geschehen kann: „ It is almost always possible to concoct a suitable response just by looking at the contents of the request packet from the client – even for complex protocols like SMB. Knowledge of prior state is not compulsory. “ ([YBP04]) Die im Paper beschriebene Struktur des „*iSink*“ wurde in zwei Umgebungen bereitgestellt: einer Campus-internen Installation, welche etwa 100.000 freie Adressen im Campus-Netz der *University of Wisconsin* umfasst, sowie einer Installation über einem *service Provider*, welche etwa 16 Millionen Adressen umfasst. Mittels des *iSink* wurden Missbrauchserscheinungen wie *Backscatter* und Würmer über einen Messzeitraum von vier Monaten untersucht.

Basierend auf *iSink* und dessen zustandsloser, skalierbarer und erweiterbarer Struktur führten Pang et. al [Pan+04] eine allgemeine Charakterisierung von Internet-Hintergrundstrahlung durch. Zusätzlich wurde eine „*LBL Sink*“ genannte Infrastruktur einge-

setzt, welche auf dem Virtuellen *Honeypot Honeyd*[Pro] basiert.
Honeyd stellt einen vollständigen, zustandsbehafteten virtuellen Honeypot bereit und ist daher nicht in gleichem Maße skalierbar wie *iSink*. Aufgrund dessen wurde der *LBL Sink* nur mit einem vergleichsweise kleinen Adressraum, welcher zehn /24 und damit etwa 2.500 IPv4 Adressen umfasst, betrieben. Für die *iSink* Messung wurde ein /8, sowie zwei /19 verwendet, sodass auch hier etwa 16 Millionen IPv4 Adressen zur Messung verwendet wurden. Die Messung wurde über einen Zeitraum von etwa drei Monaten durchgeführt.

3.1.4. Andere Ansätze

Neben den bisher beschriebenen Messverfahren, welche grundsätzlich auf dem Ansatz des *darknets*, also der Verwendung von nicht verwendeten IP-Netzwerken, basieren, wurden auch andere Messungsmethoden untersucht.

Aufgrund der Problematik, dass IPv4 Adressen stark limitiert sind und so oft nicht ausreichend große Netzwerke verfügbar sind um *darknet* Messungen durchzuführen, haben Harrop et. al. die Verwendung von *Greynets* untersucht [HA05]. Ein Greynet ist ein eigentlich genutztes IP-Netz, welches jedoch einzelne, nicht genutzte *dark hosts* enthält. Global auftretende Phänomene lassen sich damit zwar schwer analysieren, jedoch ist dieses Verfahren geeignet um Ausbrüche von Malware in eigenen Netzen detektieren zu können und ist damit beispielsweise für den Einsatz in Firmennetzen interessant.

Die *Darknet* Messung hat im Allgemeinen den Vorteil, dass auf den Adressen kein regulärer Datenverkehr stattfindet und somit keine Vorselektion zur Unterscheidung von Hintergrundstrahlung und regulärem Datenverkehr vorgenommen werden muss. Für die Analyse von Hintergrundstrahlung in einem produktiv genutzten IPv4 Netz haben Glatz et. al. ein Verfahren vorgeschlagen, welches eine Klassifikation von „*Internet One-way Traffic*“ [GD12] umsetzt. Die zugrunde liegende Idee ist, dass regulärer Datenverkehr zumeist bidirektional verläuft und unidirektionaler Datenverkehr ein Indiz für abweichendes Verhalten ist, welches Grund zur genaueren Untersuchung liefert.

3.2. Klassifikation von *Malicious Traffic*

Ausgehend von unterschiedlichen Messmethoden, unterscheiden sich auch die Möglichkeiten um den aufgezeichneten Datenverkehr in der Analyse zu klassifizieren. Je nach Art des verwendeten *responders* sind unterschiedliche Merkmale im Datensatz enthalten, welche zur Klassifizierung verwendet werden können.

3.2.1. Passive Messungen

Bei einer passiven Messung liegen für TCP-Verbindungen nur die initialen Pakete vor. Als einfaches Klassifikationsmerkmal können hier die gesetzten TCP *Flags* verwendet werden. Anhand der Flags lassen sich *Scanning* und *Backscatter* in zwei disjunkte Gruppen aufteilen.

> We classify TCP SYN packets as scanning traffic. We define backscatter traffic as TCP SYN+ACK, RST, RST+ACK, and ACK packets ([Wus+10])

Grundsätzlich wurde dieses Klassifikationsschema bei allen passiven Messungen [Wus+10]; [Czy+13]; [DBH14] verwendet.

Der von Durumeric verwendete Ansatz zur Klassifizierung von Scanning verwendet darüber hinaus Verfahren, welches die Klassifikation nicht anhand eines einzelnen Pakets, sondern für eine Menge von Paketen der selben Quelladresse vornimmt. Dabei wird eine Quelladresse als Scanner identifiziert, wenn diese mindestens 100 Adressen des Messnetzwerks auf dem gleichen Protokoll und Port anspricht. Ebenso wird eine minimale Paketrate von 10 Paketen pro Sekunde von dieser Quelladresse vorausgesetzt. Ebenso kann ein erkannter Fingerprint der Scansoftware als zusätzliches Kriterium verwendet werden. Dabei muss jedoch berücksichtigt werden, dass aufgrund der Größe des für den Fingerprint verwendeten *identification* Feldes von 16 Bit, das Paket mit einer Wahrscheinlichkeit von $\frac{1}{65536}$ fehlerhaft klassifiziert wird. Aus diesem Grund musste dieses Merkmal bei allen Paketen der Quelladresse, welche dem Scan zugeordnet wurden, übereinstimmen.

Für die Klassifikation von Backscatter Traffic verwendete Moore [Moo+06] ein dreistufiges Verfahren. Zunächst wurde der aufgezeichnete Datenverkehr auf Paketebene als Backscatter klassifiziert. Im zweiten Schritt wurden die gefilterten Pakete gruppiert. Eine Gruppe beinhaltet dabei die Pakete, die einem potentiellen Opfer eines *Denial of Service (DoS)* Angriffs zugeordnet werden. Dies wird über die IP-Adresse des Opfers gruppiert. Auf Basis von Schwellwerten für die Paketanzahl, die Angriffsdauer und die Paketrate wurden die Flows dann als *DoS*-Angriff klassifiziert.

3.2.2. Aktives Antworten

Bei Verfahren, die ein Darknet mit einer Komponente für aktives Antworten betreiben, wird zur Klassifizierung von Backscatter Datenverkehr ebenfalls ein paketbasierter Ansatz verwendet [YBP04]; [Pan+04]. Eine Unterscheidung von Backscatter `SYN ACK`

Paketen von den Pakten des *active responders* ist durch die Quelladresse möglich. Yegneswaran klassifiziert jedoch auch eingehende `TCP ACK` als Backscatter, welche innerhalb einer TCP-Verbindung auch von einem Scanner versendet werden, etwa als drittes Paket des Verbindungsaufbaus.

3.2.3. Klassifikation ohne Darknet

Die Erkennung von *Scanning* und *Backscatter* Netzwerkmitschnitten, welche nicht auf einem Darknet beruhen enthalten Datenverkehr, welcher durch die legitime Nutzung der IP-Adressen entsteht. Da hierbei *Background radiation* nicht durch die Art der Messung vorgefiltert wird, genügen einfache Klassifikationsmerkmale nicht, um die genannten Phänomene zu erkennen.

Für die Erkennung von *Backscatter* wird dabei ausgenutzt, dass eine TCP-Verbindung nicht vollständig vom Sensor erfasst wird. Bei einer *SYN Flood*, einem häufig verwendeten Verfahren für *DoS* Angriffe, empfängt der Sensor nie das initiale `SYN` Paket, sodass nur der Verbindungsaufbau in die Gegenrichtung durch das `SYN ACK` des Opfers im Mitschnitt enthalten ist. Die TCP-Verbindung erscheint in diesem also uni- statt bidirektional. [GD12].

Zur Identifizierung von *Scanning* kommen verschiedene Verfahren zum Einsatz. Eines davon basiert darauf, einen "*service fanout*" einer Absenderadresse zu messen, um diesen als Scanner zu identifizieren [GD12]; [APT07]. Ein mögliches Kriterium dazu ist die Anzahl der Tupel `(Zieladresse, Zielport)`, zu welchen der potentielle Scanner eine Verbindung aufbaut. Jedes dieser Tupel wird als *good* oder *bad* klassifiziert. Dazu wird etwa der TCP-Verbindungsstatus betrachtet, also ob die Verbindung korrekt auf- und abgebaut wurde. Das Verhältnis der *service fanouts* für gute und schlechte Verbindungen wird dann durch einen Schwellwert zur Klassifizierung des entfernten Computers als Scanner genutzt.

3.3. Probleme der Messmethodik

Bereits in den anfänglichen Arbeiten zur Messmethodik haben Moore et. al. Probleme bei der Analyse von Internet-Hintergrundstrahlung identifiziert und mögliche Lösungen vorgeschlagen [Moo+04]. Auf Grundlage dieser Lösungsansätze wurde der *Internet Motion Sensor* [Bai+05] als verteiltes *Network Telescope* implementiert und verschiedene *active responder* etabliert.

Aus dem Einsatz von aktivem Antworten in einem *Darknet* entstehen jedoch neue Probleme. Das Antworten auf Applikationsebene durch einen emulierten Host unterscheidet sich von den Antworten eines echten Servers, sodass ein Angreifer die Adressen des Darknets erkennen und aus seinem Scan ausschließen kann [Bai+06]. Auch beim aktiven Antworten auf Transportebene kann durch das Verhalten, jede eingehende Verbindungsanfrage anzunehmen, eine Identifikation des Sensors stattfinden [Bai+05].

Unabhängig vom Einsatz einer aktiven Antwortkomponente kann die Wahl der konkreten IP-Adressen für eine *Darknet-Messung* Auswirkungen auf die Messung haben. Diese als *internet address pollution* bezeichneten Auswirkungen [Wus+10] treten für konkret verwendete IP-Adressen, bzw. Subnetze auf, welche in Verbindung mit Fehlkonfigurationen oder falscher Nutzung zur "Verschmutzung" des Adressbereichs führen. Zu den Beispielen für fehlerhafte Nutzung gehört die Verwendung des Netzwerk-Testprogramms *iperf* gegen die Adresse 1.2.3.4, welche sich in dem von Wustrow verwendeten Netz 1.0.0.0/8 befindet. Eine Fehlkonfiguration, welche verschiedene Adressen des Formats 1.x.168.192 betrifft, lies sich auf eine Byte-Order-Fehlkonfiguration zurückführen. Pakete, die an Adressen des privaten Adressbereichs 192.168.0.0/16 gesendet gerichtet waren, wurden dadurch fälschlicherweise an Adressen des Darknets gesendet.

Address pollution ist ein Störfaktor, welcher nicht dem eigentlich untersuchten *background radidation* Datenverkehr entspricht, und sollte deshalb von der Messung ausgenommen werden. Die vorgeschlagene Lösung dafür ist, die betroffenen Subnetze pauschal aus der Messung zu entfernen [Wus+10]. Dies ist möglich, da das Phänomen stark geclustert in kleinen Subnetzen auftritt und nicht den gesamten beobachteten Adressraum betrifft.

4. Analyseverfahren

Der zu analysierende Datensatz ist aus einer semi-aktiven Messung hervorgegangen, welche in Abbildung 4.1 dargestellt ist. Eingehender Datenverkehr aus dem Internet wird zunächst durch einen Portfilter im Netz der Universität vorgefiltert, bevor das zur Messung verwendete Netzwerk diesen empfangen kann. Die Firewall des Messaufbaus führt keine weitere Filterung des eingehenden Datenverkehrs durch. Für ausgehenden Datenverkehr stellt diese jedoch sicher, dass ausschließlich TCP Antworten zurückgesendet werden. Andere ausgehende Pakete werden von dieser verworfen, jedoch zuvor aufgezeichnet. Eingehender TCP Datenverkehr wird von der Antwortkomponente (*Responder*) beantwortet. Der gesamte IPv4-Datenverkehr innerhalb des Messnetzwerks wird in PCAP-Dateien aufgezeichnet.

Zur Analyse dieses Datensatzes wird ein iterativer Ansatz gewählt, über welchen die vorliegenden Daten schrittweise klassifiziert und ausgewertet werden. Dazu wird jeweils ein Merkmal ausgewählt, anhand dessen die Klassifikation vorgenommen wird. Datenverkehr, welcher nicht aufgrund dieses Merkmals zugeordnet werden kann, wird in weiteren Durchläufen auf andere Merkmale hin untersucht. Die Anzahl der nicht klassifizierten Daten soll dadurch in jedem Durchlauf verringert werden.

Aufgrund vorheriger Klassifikationen von Datenverkehr, welche in Abschnitt 3.2 beschrieben wurden, sind Klassifikationsmerkmale bekannt, welche jedoch aufgrund einer abweichenden Messmethode für den vorliegenden Datensatz modifiziert werden müssen. Im Folgenden werden die wichtigsten angewendeten Klassifikationsmerkmale erläutert.

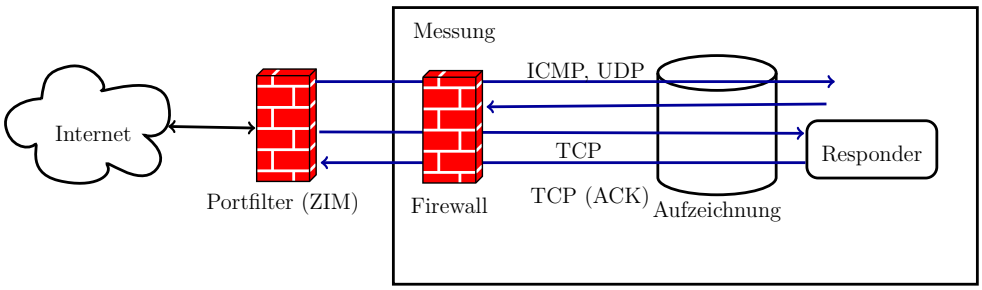


Abbildung 4.1.: Aufbau der Messung

4.1. Klassifikationsmerkmale

Welche Merkmale für die Klassifikation eines Datenpakets relevant sind, ist vor allem von den verwendeten Protokollen abhängig. Für die verbreiteten Transportprotokolle TCP und UDP, sowie das Diagnoseprotokoll ICMP müssen jeweils andere Verfahren angewendet werden.

4.1.1. ICMP

ICMP-Pakete lassen sich anhand des Typs, welcher durch das Protokoll als eine Kombination der zwei Felder `Type` und `Code` vorgegeben wird, klassifizieren. Ein `Echo Request`, auch Ping genannt, ist als Erreichbarkeits-Scan zu klassifizieren, ein `Echo Reply` als Backscatter, da das Messnetzwerk selbst keine Requests versendet und der Reply so nur aufgrund einer gefälschten IP-Adresse zum Messnetzwerk gesendet werden kann.

Die verschiedenen Codes des Typen `Destination unreachable` können bei einer rein passiven Messung als Backscatter klassifiziert werden. Bei einer semi-aktiven Messung werden jedoch auch TCP *Acknowledgements* vom Messnetzwerk verschickt. Bei der Auswertung muss daher betrachtet werden, ob die empfangenen ICMP-Pakete eine Reaktion auf diese Antwortpakete sind. Dies ist aufgrund des Datensegments im ICMP-Paket möglich. Dieses enthält das IP Paket, welches das Versenden der ICMP-Nachricht ausgelöst hat. Nur wenn dort kein vom Messnetzwerk versendetes TCP-Paket enthalten ist, handelt es sich bei dem ICMP-Paket um Backscatter. Andernfalls ist dieses ein Artefakt der Messmethodik. Eine Auflistung des Klassifikationsschemas für ICMP-Pakete ist in Tabelle 4.1 dargestellt.

Type	**active responder Paket in Payload**	**Klasse**
* (alle)	Ja	Artefakt
0 (Echo Antwort)	Nein	Backscatter
3 (Ziel nicht erreichbar)	Nein	Backscatter
8 (Echo Anfrage)	Nein	Scan
11 (Zeit Abgelaufen)	Nein	Backscatter
14 (Zeitstempel Antwort)	Nein	Backscatter

Tabelle 4.1.: Klassifikation von ICMP-Nachrichten

4.1.2. UDP

Der Header eines UDP-Datagramms enthält im Allgemeinen keine verlässlichen Informationen zur Klassifizierung des Datenverkehrs. Quell- und Zielport können, falls es sich um *well-known* Ports handelt zwar ein Indiz auf die Art des Datenverkehrs geben,

jedoch sind die Ports kein hinreichendes Kriterium zur Identifikation des verwendeten Anwendungsprotokolls. UDP-Datenverkehr wird daher anhand der übertragenen Daten des Anwendungsprotokolls untersucht.

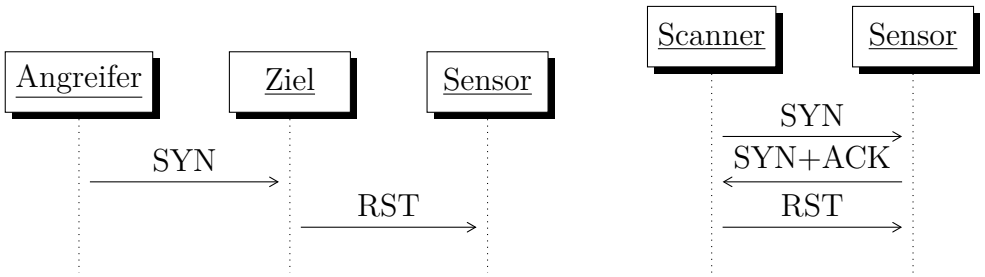
Da das semi-aktive Messnetzwerk selbst keine UDP-Datagramme versendet, lassen sich auf einem einfachen Frage-Antwort-Mechanismus basierende Protokolle wie DNS auf diese Art und Weise klassifizieren. Eine DNS-Anfrage ist ein Merkmal für einen Scanner, welcher etwa zum Auffinden von *Open Resolvern* eingesetzt wird. Im Gegensatz dazu lässt sich eine DNS-Antwort der Kategorie Backscatter zuordnen.

4.1.3. TCP

Zur Klassifikation von TCP-Datenverkehr existieren in Abhängigkeit der verwendeten Messmethodik verschiedene Verfahren. Bei Darknet-Messungen erfolgt die Klassifikation zumeist anhand der TCP-Flags. Bei Einsatz eines *active Responders* ist dies jedoch nicht möglich, da durch das Versenden von TCP-Acknowledgements bestimmte Flags in mehreren Situationen auftreten können.

Abbildung 4.2 zeigt ein Beispiel dafür, in welchen Fällen ein TCP-Paket mit gesetztem `RST`-Flag auftreten kann.. Dieses Flag signalisiert ein sofortiges Ende der Verbindung. Bei einem Darknet wird dies als Backscatter klassifiziert, da dies als Antwort eines Servers gesehen wird, welcher aufgrund eines DoS-Angriffs keine weiteren Verbindungsanfragen mehr entgegennehmen kann (a). Bei einer semi-aktiven Messung ist ein weiterer Fall möglich: Der Scanner sendet ein `SYN`-Paket, welches mit einem `SYN+ACK`-Paket beantwortet wird. Bei einem protokollkonformen Verbindungsaufbau würde der Scanner an dieser Stelle ein weiteres `ACK`-Paket versenden, welches den Verbindungsaufbau abschließt. Stattdessen sendet er ein `RST`-Paket um die Verbindung zu beenden. Dieses als *Stealth Scan* (b) bezeichnete Verfahren benutzt der Scanner um möglichst unentdeckt zu bleiben. Die zugrunde liegende Idee ist, dass der Verbindungsaufbau auf diese Weise nicht abgeschlossen ist, und die TCP-Implementierung des Servers daher dem darüber liegenden Anwendungsprotokoll noch keine neue Verbindung signalisiert hat. Auf diese Weise erscheint der Scanner nicht in den Log-Dateien des Servers. Dennoch kann der Scanner aufgrund des `SYN+ACK`-Pakets, bzw. dessen Ausbleiben erkennen, ob der gescannte Port offen ist oder nicht.

Für semi-aktive Messungen ist ein paketbasierter Klassifizierungsansatz daher ungeeignet. Alternativ ist es möglich die Klassifizierung auf Basis einer zustandsbasierten Verbindungserkennung durchzuführen. Dazu werden zunächst die TCP-Verbindungen, welche im Datensatz als einzelne Pakete aufgezeichnet wurden, wieder zu einer Verbindung



(a) als Backscatter (b) durch Stealth Scan

Abbildung 4.2.: Mögliches Auftreten von RST-Paketen bei semi-aktiven Messungen

zusammenzusetzen. Durch den Kontext der Verbindung lässt sich dann für den zuvor beschriebenen Fall entscheiden, ob das `RST`-Paket als Teil eines Stealth-Scans oder durch Backscatter entstanden ist.

Für die Klassifikation der TCP-Verbindung wird im Rahmen dieser Arbeit folgender Ansatz gewählt:
Am Anfang einer Verbindung muss immer ein `SYN`-Paket gesendet werden. Existiert dieses nicht in einer zusammengesetzten Verbindung, kann dies nur aufgrund einer gefälschten IP-Adresse aufgezeichnet worden sein, da das Messnetzwerk selbst keine Verbindungen initiiert. Solche Verbindungen werden daher als Backscatter klassifiziert.
Ein Scan lässt sich unter anderem durch die bereits beschriebene Methodik des *Stealth Scan* identifizieren. Das entscheidende Kriterium hierbei ist, dass zwar das `SYN`-, sowie das `SYN+ACK`-Paket des Verbindungsaufbaus vorhanden sind, dieser jedoch nicht durch das `ACK`-Paket abgeschlossen wurde.

Eine weitere Klassifizierungsmethode basiert auf dem von Durumeric [DBH14] benannten Verfahren zum Erkennen von Scan-Software anhand eines digitalen Fingerabdrucks. Möglich ist dies für die Tools Masscan und Zmap. Das `identification`-Feld des IP-Headers, in welchem sich dieser Fingerabdruck befindet, hat eine Größe von 16 Bit, sodass $2^{16} = 65536$ verschiedene Werte möglich sind. Dies hat zur Folge, dass bei einem einzelnen Paket, welches nicht von der jeweiligen Scan-Software versendet wurde, in durchschnittlich einem von 65536 Fällen das Paket zufällig diesen Wert haben kann. Das entspricht einer Wahrscheinlichkeit von 0,0015%. Die Wahrscheinlichkeit einer fehlerhaften Klassifikation über dieses Merkmal verringert sich stark bei der Betrachtung einer TCP-Verbindung, wenn dort mehrere Pakete des Clients vorliegen. Entsprechend der Anzahl N der empfangenen Pakete in der Verbindung erfolgt eine Fehlklassifikation mit der Wahrscheinlichkeit $\frac{1}{2^{N \cdot 16}}$.

Mittels dieser Klassifizierungsmerkmale können nur Teile des TCP-Datenverkehrs zugeordnet werden. Wird eine andere Scanner-Software eingesetzt kann diese nicht aufgrund

der genannten Merkmale identifiziert werden. Ebenso werden Scans, welche den Verbindungsaufbau vollständig durchführen, nicht erkannt. Daher erfolgt wie bei UDP eine weitere Analyse anhand der übertragenen Daten im Payload der Verbindungen. Wenn kein Payload in einer Verbindung vorhanden ist, kann dies zwei Ursachen haben. Die erste Möglichkeit ist, dass das Anwendungsprotokoll einen eigenen Verbindungsaufbau vorschreibt, bei welchem der Server die initiale Nachricht versendet. Da bei der vorliegenden Messung das Verhalten von Anwendungsprotokollen nicht simuliert wird, erfolgt dieser Handshake nicht und der Client wird die Verbindung nach Ablauf eines Timeouts beenden. Falls das Anwendungsprotokoll keinen vom Server initiierten Verbindungsaufbau vorschreibt, werden vom Client die ersten Daten auf der etablierten Verbindung versendet. Schließt der Client jedoch die Verbindung ohne Daten versendet zu haben, deutet dies auf einen Port-Scan hin, bei welchem der Scanner lediglich feststellen möchte, ob der Port offen ist oder nicht. Als Port-Scan werden hier also Verbindungen klassifiziert, auf welchen keine Daten übertragen wurden und deren Dauer unterhalb eines Schwellwert liegt, welcher zur Differenzierung von ausbleibenden Server Nachrichten dient. Die Wahl des Schwellwerts hat dabei Auswirkungen auf die Erkennungsrate. Ist der Wert zu niedrig gewählt, werden Port-Scans nicht klassifiziert, wenn diese die Verbindung zu spät schließen. Ist der Wert jedoch zu hoch gewählt, werden Verbindungen fälschlicherweise als Port-Scans klassifiziert, obwohl der Client noch auf eine Nachricht des Servers wartet. Bei dieser Analyse wird der Wert auf zehn Sekunden festgelegt.

Aus einer vorherigen Analyse von Teilen des vorliegenden Datensatzes aus dem Jahr 2013 [Sch13] ist bekannt, dass in den Aufzeichnungen auch BitTorrent Datenverkehr enthalten ist, welcher sich auf eine fehlerhafte Implementierung der Bibliothek `libtorrent` [libtorrent] zurückführen lässt. Das *Filesharing* Protokoll BitTorrent hat keinen fest zugeordneten Port, sondern sieht eine dynamische Portauswahl vor. Als Standardbereich werden dafür zwar die Ports 6881 – 6889 verwendet, jedoch ist grundsätzlich auch die Nutzung anderer Ports möglich. Eine Erkennung kann daher am Besten aufgrund des Payloads durchgeführt werden. Ein BitTorrent Verbindungsaufbau lässt sich über eine feste Sequenz (`0x13426974546f7272656e742070726f746f636f6c`) zu Beginn jeder Verbindung klassifizieren. Dies entspricht dem Byte mit dem Zahlenwert 19, gefolgt von der Zeichenkette “BitTorrent protocol”. Dies ist jedoch nur bei unverschlüsseltem BitTorrent-Datenverkehr möglich.

4.2. Iterationsverfahren

Um den Datensatz zu analysieren wird in einer ersten Iteration zunächst der Datenverkehr extrahiert, welcher keines der drei Protokolle TCP, UDP und ICMP, welche genauer

betrachtet werden sollen, beinhaltet. Im Anschluss daran wird jeweils der Datenverkehr für diese drei Protokolle getrennt betrachtet.

Da für ICMP ein vollständiges Klassifikationsschema (Tabelle 4.1) vorliegt, kann dieser direkt klassifiziert werden und muss nicht durch mehrere Iterationen schrittweise aufgeschlüsselt werden.

Die Analyse des UDP-Datenverkehrs wird in mehreren Iterationen vorgenommen. Eine Iteration besteht dabei aus drei Phasen. In der ersten Phase werden die Daten anhand verschiedener Kriterien gezählt. Diese Kriterien sind der Quell- und der Zielport des UDP-Datagramms, sowie die ersten Bytes des übertragenen Payloads. Erfasst werden dabei sowohl die Anzahl der Pakete, als auch die Summe der übertragenen Bytes. In der zweiten Phase werden anhand dieser Zählung die häufigsten Vertreter der jeweiligen Merkmale bestimmt und weiter ausgewertet. Bei dieser Auswertung werden die Ursachen der prägnanten Merkmale im Detail untersucht. Für die dritte Phase der Iteration werden die untersuchten Merkmale aus dem Datensatz extrahiert. Auf dem so reduzierten Datensatz wird dann eine erneute Iteration nach dem gleichen Schema durchgeführt.

Zur Analyse des TCP-Datenverkehrs wird ebenso dieses iterative Verfahren eingesetzt, wobei hier zusammengesetzte Verbindungen anstelle einzelner Pakete betrachtet werden. Für bekannte Phänome wie Backscatter, Scanning oder BitTorrent lässt sich die erste Phase der Iteration vereinfachen, indem die in Abschnitt 4.1 beschriebenen Klassifikationsmerkmale genutzt werden. Anstelle markanter Ports und Payloads können diese Merkmale zur Identifikation und Extraktion der Daten verwendet werden. Die ersten vier Iterationen werden daher anhand der Merkmale dieser Phänomene durchgeführt. Für die anschließende Analyse des übrigen Datenverkehrs werden die Iterationen mit den drei ursprünglich beschriebenen Phasen durchgeführt.

4.3. Vorbereitung der Analyse

Der vorliegende Datensatz enthält den gesamten Datenverkehr, welcher an das verwendete semi-aktive Messnetzwerk (Abbildung 4.1) gesendet wurde, sowie dessen Antworten auf eingehende Anfragen, welche vom Betriebssystem sowie dem *active responder* versendet wurden. Dies umfasst auch ICMP-Pakete, welche das Betriebssystem als Antwort auf eingehende *echo requests*, bzw. auf Datagramme zu geschlossenen UDP-Ports versendet. Durch den Einsatz einer Firewall zwischen dem Messnetzwerk und dem Internet werden solche Pakete jedoch herausgefiltert. Die Firewall ist so konfiguriert, dass nur ausgehende TCP-Pakete mit gesetztem `ACK`-Flag versendet werden können. Eingehende

Pakete werden an dieser Stelle nicht gefiltert. Das Zentrum für Informations- und Mediendienste (ZIM) der Universität, welches dem Lehrstuhl das Netz zur Verfügung stellt, betrieb jedoch bis zum Ende der Messung einen Port-Filter, wodurch Pakete gefiltert wurden. Anhang A enthält eine Auflistung der Filter, welche im März 2017 aktiv waren. Angaben zum genauen Zeitpunkt der Einrichtung einzelner Filterregeln liegen nicht vor.

Da der ausgehende Datenverkehr für ICMP und UDP-Pakete von einer Firewall gefiltert wird und daher keine Kommunikation mit anderen Hosts im Internet ermöglicht, wird dieser in der Analyse ebenfalls gefiltert. Der ausgehende TCP-Datenverkehr ist jedoch für die Rekonstruktion der Verbindungen relevant und unterliegt daher keiner Filterung.

Um eine effiziente Verarbeitung des Datensatzes zu ermöglichen, sollen die aufgezeichneten Daten parallel verarbeitet werden. Die Daten liegen initial als 25 PCAP-Dateien vor, welche in chronologischer Reihenfolge benannt sind und jeweils disjunkte Zeitabschnitte beinhalten. Aufgrund der starken Schwankung bei der Größe der Dateien – zwischen 74MB und 49GB – lässt sich damit jedoch keine gleichmäßige Auslastung mit vielen parallelen Threads erzielen. Weiterhin beträgt der Abstand zwischen jeweils zwei aufeinander folgenden Dateien teilweise nur wenige Sekunden. Die Rekonstruktion von TCP-Verbindungen müsste daher über die Dateigrenzen hinaus durchgeführt werden.
Um eine bessere Parallelisierbarkeit sowie eine Vereinfachung der TCP-Rekonstruktion zu erreichen, werden die Dateien in der Vorverarbeitung umgeschrieben. Dazu werden 95 PCAP-Dateien erzeugt, welche jeweils den Datenverkehr einer Zieladresse des Messnetzwerks beinhalten. So ist sichergestellt, dass sich alle Pakete einer TCP-Verbindung innerhalb einer Datei befinden und die Rekonstruktion vollständig auf der Dateiebene parallelisiert werden kann. Die hierbei resultierenden Dateien sind ebenfalls unterschiedlich groß, jedoch ist die Schwankung deutlich geringer als zwischen den ursprünglichen Dateien. Vor der Verarbeitung war die größte Datei um den Faktor 678 größer war als die Kleinste. Durch die Vorverarbeitung wurde dieser Wert auf 46 reduziert.

Die Rekonstruktion der TCP-Verbindungen erfordert zwei Teilschritte. Zunächst müssen fragmentierte IP-Pakete zusammengesetzt werden. Fragmente, welche zum gleichen Paket gehören, werden durch das Tripel `(Quell-IP, Ziel-IP, Identification)` identifiziert. Die Reihenfolge der Fragmente ergibt sich aus dem `Offset`-Feld im Header des IP-Fragments.
Eine TCP-Verbindung kann in eine Richtung durch das Quadrupel `(Quell-IP, Ziel-IP, Quell-Port, Ziel-Port)` identifiziert werden. Die Identifikation der Gegenrichtung erfolgt durch paarweises Vertauschen der beiden IP-Adressen und der beiden Ports. Um zu überprüfen, ob ein Paket zu einer bereits offenen TCP-Verbindung gehört, müssen entsprechend beide Quadrupel getestet werden. Die über eine TCP-Verbindung übertra-

genen Nutzdaten können durch mehrere Pakete verschickt worden sein. Teil der Rekonstruktion ist daher, diese möglicherweise in falscher Reihenfolge erhaltenen Pakete korrekt anzuordnen. Dies erfolgt auf Grundlage der Sequenznummer, welche aufsteigend für die Pakete in jede Richtung vergeben wird.

4.4. Aufbau der Analysesoftware

Die Konzeption der Analysesoftware orientiert sich am Software-Entwurfsmuster *pipes-and-filters*. In diesem Modell existieren Filter mit je einem Eingang und einem Ausgang. Jeder Filter führt eine definierte Transformation der Daten am Eingang aus und schreibt diese in den Ausgang. Durch die Verwendung von Pipes wird der Ausgang eines Filters mit dem Eingang eines Anderen verknüpft. Dies ermöglicht die flexible Kombination mehrerer Filter zu einem Gesamtsystem. Im Folgenden wird für die im Entwurfsmuster "Filter" genannten Elemente der Begriff "Komponente" verwendet. Ein Grund dafür ist, dass Komponenten in dieser Software mehrere Ausgänge haben können, was in *pipes-and-filters* nicht vorgesehen ist. Weiterhin wird der Begriff "Filter" für eine spezielle Komponente zur Filterung von Daten verwendet.

Nach dem Konzept einer solchen Pipeline werden für die Software verschiedene Komponenten geschaffen, welche miteinander verbunden werden können. Jede Komponente hat einen Eingang sowie abhängig von ihrer Art keinen bis zwei Ausgänge. Ein möglicher Aufbau einer solchen Pipeline wird in Abbildung 4.3 dargestellt. Der Dateileser liest den Inhalt der PCAP-Datei aus und leitet die noch ungeparsten Daten an das Parser-Element weiter. In diesem werden die verwendeten Protokolle des aufgezeichneten Pakets identifiziert. Das mit diesen Informationen angereicherte Paket wird dann an einen Filter gesendet, welcher prüft, ob es sich um ein gesendetes oder ein empfangenes Paket handelt. Abhängig davon wird das Paket durch einen der beiden Ausgänge an eine Zählkomponente geleitet, welche die verschiedenen Attribute wie etwa die Paketanzahl oder die Größe der Pakete zusammenzählt. Der abschließende Endpunkt wird aufgrund einer technischen Notwendigkeit am Ende angefügt.

Für diese Verarbeitung werden Transformationen der Daten vorgenommen, sodass Komponenten nicht beliebig verkettet werden können. Der im Beispiel gezeigte Filter benötigt bereits als Paket geparste Daten und kann daher nicht direkt mit der Dateieingabe verknüpft werden.
Die in der Software vorkommenden Datenformate sind die ungeparsten Aufzeichnungsdaten (*UnparsedMeasurement*), die geparsten Aufzeichnungsdaten (*Measurement*) und rekonstruierte TCP-Verbindungen (*TCPStream*). Die ersten beiden bestehen jeweils aus

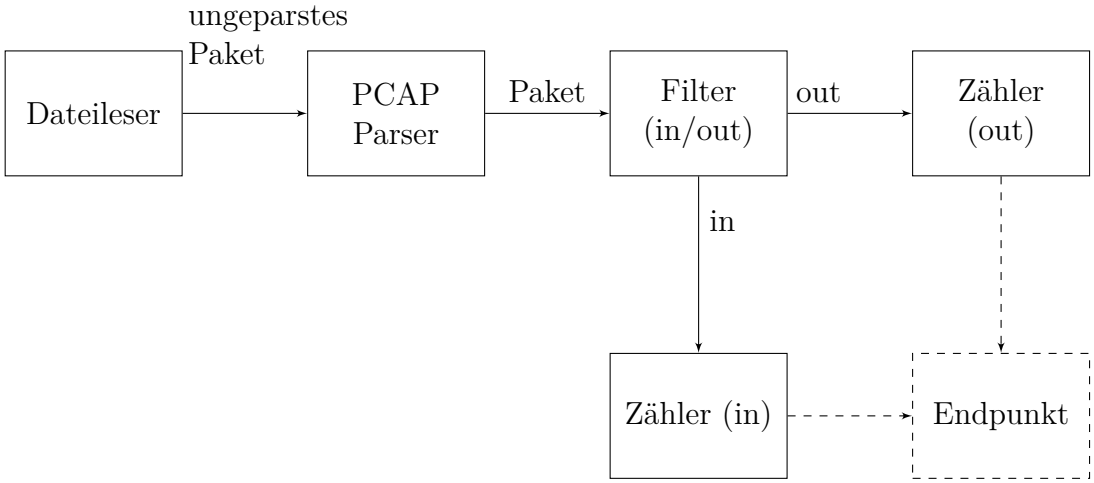


Abbildung 4.3.: Exemplarischer Aufbau einer Pipeline zum Zählen des eingehenden und ausgehenden Datenverkehrs

einem Tupel `(CaptureInfo, []byte)`, bzw. `(CaptureInfo, Packet)`. Die CaptureInfo beinhaltet dabei Metadaten zum aufgezeichneten Record, wie sie im PCAP-Datenformat für jedes Paket spezifiziert sind. Eine rekonstruierte TCP-Verbindung besteht aus diversen gesammelten Daten wie dem Anfangs- und Endzeitpunkt, dem übertragenen Payload und Informationen zu verwendeten Steuerdaten.

Konfiguration Um die Pipeline-Komponenten für jede Iteration der Analyse flexibel kombinieren und konfigurieren zu können, kommt eine XML-Konfigurationsdatei zum Einsatz. In dieser werden sowohl die Parameter einzelner Komponenten als auch die Verknüpfung der Komponenten untereinander festgelegt. Durch die Fixierung des Analyseaufbaus in einer Konfigurationsdatei ist weiterhin der Analyseaufbau dokumentiert und auf zukünftige Messungen erneut anwendbar.

Parallelisierung Die Pipeline-Architektur ermöglicht ebenfalls eine einfache Parallelisierung der Analyse auf zwei Ebenen. Nachdem eine Komponente die Bearbeitung eines Pakets abgeschlossen und dieses innerhalb der Pipeline weitergegeben hat, kann bereits das nächste Paket verarbeitet werden. Dies kann gleichzeitig zur Verarbeitung des vorherigen Pakets durch die nächte Komponente erfolgen. Ebenso kann eine Parallelisierung erfolgen, indem mehrere Dateien gleichzeitig verarbeitet werden. Dazu wird die Pipeline entsprechend oft dupliziert.

5. Implementierung

Für die Verarbeitung von Internet-Datenverkehr gibt es verschiedene etablierte Softwarelösungen. Dazu gehören die Kommandozeilenanwendung `tcpdump`[1], welches über die Bibliothek `libpcap` für die Aufzeichnung der Daten im PCAP-Datenformat (Abschnitt 2.5) ermöglicht. Eine ebenfalls verbreitete Software ist `Wireshark`[2], welches über eine grafische Benutzeroberfläche die aufgezeichneten Daten aufbereitet und Analyseverfahren für gängige Protokolle bereitstellt. Beide Programme sind jedoch für das hier angewendete Analyseverfahren nur bedingt geeignet.

`tcpdump` ist primär als Werkzeug zum Ausgeben und Aufzeichnen des Datenverkehrs konzipiert und bietet wenig Möglichkeit zur Analyse der Daten. `Wireshark` hingegen ermöglicht zwar mehr Analysemöglichkeiten und lässt sich über die Skriptsprache Lua leicht erweitern, ist allerdings nicht für sehr große Datenmengen konzipiert, da die gesamte PCAP-Datei gleichzeitig in den Arbeitsspeicher geladen wird. Bei einem Gesamtumfang der aufgezeichneten Daten von etwa 238 GB ist dies auf der verfügbaren Hardware nicht möglich.

Für die Durchführung der angedachten Analyse wird daher eine Software benötigt, welche die vorhandenen Datenmengen effizient mit den zur Verfügung stehenden Ressourcen verarbeiten kann. Weiterhin soll die Software um neue Analysekomponenten erweiterbar und auch ohne Anpassung des Quellcodes stark konfigurierbar sein. Für die Implementierung dieser Software wird die Programmiersprache Go[3] eingesetzt.

Go ist eine stark typisierte Programmiersprache, welche von Google mit Fokus auf hohe Performance sowie die Unterstützung nebenläufiger Programmierung entwickelt wurde. Funktionsaufrufe in Go können durch das Voranstellen des Befehls `go` asynchron gestartet werden. Man spricht dann von *Go-Routinen*. Zur Synchronisation der Routinen orientiert sich Go am Konzept *share memory by communication* und bietet dazu das Sprachkonstrukt des Channels. Ein Channel ist ein Kanal, in den Prozesse Werte hineinschreiben und diese auslesen können. Lesezugriffe auf einen leeren, sowie Schreibzugriffe

[1] `http://www.tcpdump.org/tcpdump_man.html`
[2] `https://www.wireshark.org/`
[3] `https://golang.org/`

auf einen vollen Channel blockieren, sodass mehrere Routinen hierdurch synchronisieren können. Außerdem sind Channel typisiert. Go kompiliert zu Maschinencode und verwendet einen Garbage Collector zur Speicherbereinigung.

Für die Verarbeitung der PCAP-Dateien wird die Bibliothek `gopacket`[4] verwendet. Diese bietet verschiedene Module, welche in der Implementierung der Analysesoftware eingesetzt werden. Dazu gehören der Parser für PCAP-Dateien, ein Parser für gängige Netzwerkprotokolle sowie Funktionalität zum Defragmentieren von IP-Paketen und Rekonstruieren von TCP-Verbindungen.

5.1. Pipeline-Elemente

Die erste Komponente jeder Pipeline ist der *Fileworker*. Dieser liest mit Hilfe der Bibliothek `gopacket` die PCAP-Records aus und leitet diese als *Unparsed Measurement* an die nächste Komponente weiter.
Zum Auslesen der PCAP-Dateien bietet `gopacket` zwei mögliche Implementierungen an, einen Wrapper um die C-Bibliothek `libpcap` sowie eine in Go geschriebene Implementierung, welche auf dem Reader-Interface der Standard-Bibliothek aufbaut. Die libpcap-Implementierung unterliegt dadurch diversen Einschränkungen. Als Datenquelle kann lediglich ein Dateipfad angegeben werden, aus welchem die Daten gelesen werden. In der in Go geschriebenen Implementierung können hingegen die Vorteile des Reader-Interfaces genutzt werden. Dazu gehört die Möglichkeit verschiedene Reader ineinander zu verschachteln. Dies ermöglicht den einfachen Einbau eines Lesepuffers, sodass anstatt vieler kleiner Leseoperationen auf der Festplatte immer große Datenblöcke ausgelesen und im Arbeitsspeicher gehalten werden. Das Parsen der PCAP-Dateien konnte durch den Einbau des Puffers signifikant beschleunigt werden. Demnach wird für die Analysesoftware die in Go geschriebene Implementierung verwendet.

Als zweite Komponente jeder Pipeline wird der *Parser* verwendet. Dieser verarbeitet unter Verwendung von `gopacket` die ausgelesenen Daten der PCAP-Datei, die bisher als Byte-Array vorliegen, zu `Packet`-Objekten, welche ein abstraktes Interface für den Zugriff auf die verschiedenen Protokollschichten, sowie deren spezifische Information bieten. Die entstandenen Pakete können dann durch weitere Komponenten ausgewertet werden. Zur Durchführung dieses Vorgangs unterstützt der *Parser* verschiedene Betriebsmodi. Unterschieden wird dabei zwischen den Optionen *Lazy* und *Eager*, welche von `gopacket` zur Verfügung gestellt werden. Bei der Verwendung des *Lazy*-Modus werden die einzelnen Protokollschichten erst verarbeitet, wenn diese ausgelesen werden sollen. Dadurch

[4] `https://godoc.org/github.com/google/gopacket`

kann die für das Parsen benötigte Rechenzeit reduziert werden, da unbenötigte Protokollschichten nicht verarbeitet werden müssen. Dies bedeutet jedoch auch, dass die abgerufenen Informationen bei Abruf möglicherweise noch ausgewertet werden müssen. Im *Eager*-Modus werden zu Beginn alle vorhandenen Protokollschichten ausgewertet. Die Informationen stehen dadurch bei Abruf direkt zur Verfügung, werden jedoch auch verarbeitet, wenn sie nicht benötigt werden. Da bei einzelnen Analysedurchläufen meist nur einzelne Protokollschichten ausgewertet werden, wird hier der *Lazy*-Modus verwendet.

Paketbasierte Pipeline-Komponenten Zur Verarbeitung von Paketen stehen die Komponenten *PacketFilter*, *PacketCounter* und *PacketOutput* zur Verfügung. Der *PacketFilter* wendet auf jedes eingehende Paket Filterkriterien an. Abhängig davon, ob diese Kriterien zutreffen wird das Paket entweder über den ersten oder den zweiten der beiden Ausgänge weitergeleitet. Als Grundlage für Filterkriterien können sowohl der Zeitpunkt, an welchem das Paket empfangen wurde, als auch der Inhalt des Pakets verwendet werden. Zur Prüfung des Paketinhalts kommen *Berkley Packet Filter (BPF)*[5] zum Einsatz. Das Filterkriterium wird als Zeichenkette definiert, welche in einen binären Filter kompiliert wird, welcher auf das Paket angewendet wird. Eine Filterung des UDP-Datenverkehrs lässt sich etwa durch den Filter `ip proto udp` erreichen. Als Implementierung für BPF wird erneut `gopacket` verwendet, welches hierzu die `libpcap` benutzt.

Um die Anzahl von Paketen, welche durch die Verwendung von Filtern aufgeteilt wurden, zu ermitteln kommen *PacketCounter* zum Einsatz. Jeder *PacketCounter* enthält dazu einen oder mehrere Akkumulatoren. Ein Akkumulator ist ein zweidimensionaler assoziativer Speicher, welcher ein Paar von Zeichenketten zu einer Ganzzahl abbildet (`(string, string) -> int`). In einer Dimension werden verschiedene Werte zu den Paketen gezählt, etwa deren Anzahl oder deren Größe. In der zweiten Dimension werden die Pakete zuvor nach einem Kriterium klassifiziert. Diese Klassifizierung ermöglicht beispielsweise getrennte Zählvorgänge nach TCP-Port, Zeitraum oder IP-Adresse. Ein *PacketCounter* mit mehreren Akkumulatoren wählt den jeweils zu Verwendenden anhand eines zweiten Klassifikators aus. Dadurch wird ermöglichst, den Zählvorgang über mehrere Kriterien aufzuteilen. Ein Beispiel hierfür wäre die Aufteilung der Paketanzahl je Port, welche zusätzlich für jedes Jahr der Messung aufgeschlüsselt ist. Am Ende der Analyse wird der Inhalt jedes Akkumulators in eine CSV-Datei exportiert.

Der *PacketOutput* ermöglicht den Export von Paketen in neue PCAP-Dateien, etwa um vorgefilterte Pakete einer zusätzlichen Analyse zu unterziehen. Der *PacketOutput* unter-

[5] `https://biot.com/capstats/bpf.html`

stützt dabei wie der *PacketCounter* die Verwendung von Klassifikatoren, um die Daten in mehrere Dateien aufzuteilen. Die Erzeugung dieser vorgefilterten Dateien ermöglicht die Analyse einzelner Phänomene mittels anderer Software wie `Wireshark`, welche zwar für die Verarbeitung großer Datenmengen ungeeignet ist, jedoch die Analyse kleinerer, vorgefilterter Daten vereinfacht.

Rekonstruktion von TCP-Verbindungen Für die Verarbeitung von TCP-Verbindungen müssen diese zunächst aus den vorliegenden einzelnen Paketen zusammengesetzt werden. Die *Composer*-Komponente führt diese Transformation durch. Der *Composer* hält dazu eine Sammlung noch nicht abgeschlossener TCP-Verbindungen. Bei der Verarbeitung jedes Pakets wird geprüft, ob sich dieses zu einer der momentan offenen Verbindungen zuordnen lässt. Ist dies der Fall, wird es entsprechend zugeordnet. Andernfalls wird eine neue Verbindung auf Grundlage dieses Pakets geöffnet. Eine Verbindung wird als abgeschlossen angesehen, sobald ein eingehendes Paket das Ende der Verbindung signalisiert. Dies ist durch ein gesetztes `FIN` oder `RST` Flag möglich. Verbindungen, welchen kein solches Paket zugeordnet werden kann, werden mittels eines *Sliding-Window*-Verfahrens abgeschlossen. Das Zeitfenster wird in regelmäßigen Schritten verschoben. Verbindungen, deren letztes Paket vor Beginn dieses Zeitfensters empfangen wurde, werden als abgeschlossen angesehen.

Ebenfalls führt der *Composer* die Defragmentierung von IP-Paketen durch. Eine spezielle Konfigurationsoption ermöglicht weiterhin, dass nur die Defragmentierung durchgeführt wird, falls diese für eine paketbasierte Analyse benötigt wird. Für die Defragmentierung kommt das Modul `ip4defrag` und für die Zusammensetzung der TCP-Verbindungen das Modul `reassembly` der Bibliothek `gopacket` zum Einsatz. Während der Implementierung wurden jedoch Fehler innerhalb dieser Module festgestellt, welche im Rahmen dieser Arbeit behoben wurden[6].

Fehlerkorrekturen in gopacket Ein Fehler betrifft das Modul `ip4defrag`. Dieses beinhaltet die Funktionalität unvollständige Pakete, für welche keine weiteren Fragmente empfangen wurden, zu verwerfen. Der Zeitpunkt, zu welchem das letzte Fragment eines Pakets eingetroffen ist, wurde jedoch immer auf den Ausführungszeitpunkt der Bearbeitung gesetzt. Bei der Auswertung von Live-Mitschnitten eines Netzwerk-Interfaces ist dies eine ausreichende Näherung für diesen Anwendungsfall. Für die Verarbeitung von Netzwerkaufzeichnungen ist dies jedoch ungeeignet, da der Ausführungszeitpunkt des Programms zu weit vom Aufzeichnungszeitpunkt entfernt ist. Konsequenz dessen

[6] Die vorgenommenen Änderungen wurden den Entwicklern von `gopacket` als GitHub Pull Request zur Verfügung gestellt.

ist, dass bei der Verarbeitung von aufgezeichneten Daten kein *Sliding-Window*-Verfahren zum Verwerfen unvollständiger IP-Pakete möglich ist und diese zunehmend viel Arbeitsspeicher benötigen. `ip4defrag` wurde daher um die Möglichkeit erweitert den Aufzeichnungszeitpunkt eines Fragments in den Verarbeitungsprozess mit einzubeziehen.

Auch in dem zur Rekonstruktion von TCP-Verbindungen verwendeten Modul `gopacket` waren Anpassungen des Quellcodes notwendig. Wird eine offene Verbindung fertiggestellt, etwa durch ein empfangenes `FIN`, `RST` oder da die Verbindung sich außerhalb des *Sliding-Windows* für aktive Verbindungen befindet, ruft `gopacket` eine Callback Methode `ReassemblyComplete` auf dem Verbindungsobjekt auf, welche einen `boolean` zurückgibt. Die Rückgabe von `true` wird dabei so interpretiert, dass die Verbindung endgültig aus dem Pool der offenen Verbindungen entfernt werden soll. Wird `false` zurückgegeben, wird die Verbindung noch beibehalten. Dies ermöglicht, dass nach Abschluss der Verbindung eintreffende Pakete, etwa ein `FIN+ACK`, mit welchem die Gegenseite den Verbindungsabbau bestätigt, noch im Nachhinein verarbeitet werden können.

Die Möglichkeit, die Verbindungen wie beschrieben offen zu halten, führte jedoch zu verschiedenen Fehlern. Zum einen ordnete `gopacket` auch nach Ende der Verbindung empfangene SYN-Pakete fälschlicherweise der alten Verbindung zu, anstatt eine neue zu öffnen. Dadurch wurden weniger Verbindungen erkannt und die Zusammensetzung der übertragenen Daten war fehlerhaft. Weiterhin ist dieses Verfahren nicht mit der Pipeline-Struktur der Analyse-Software kompatibel. Die Software geht davon aus, dass jeweils immer nur eine Komponente Operationen an einer Verbindung ausführt. Dadurch kann auf zusätzliche Lock-Mechanismen zur Vermeidung von Race-Conditions verzichtet werden. die Callback-Methode `ReassemblyComplete` gibt die Verbindung an die nächste Pipeline-Komponente weiter. Bleibt die Verbindung offen, kann diese jedoch noch modifiziert werden, während die nächste Komponente bereits die Daten ausliest. Daher werden Verbindungen, für welche das Callback aufgerufen wird, endgültig geschlossen.

Das Modul `gopacket` nutzt einen `StreamPool`, welcher offene Verbindungen verwaltet und ein eigenes Speichermanagement beinhaltet. Durch einen noch unbekannten Fehler im Kontrollfluss, von `gopacket` wird die Funktion `(p *StreamPool) remove(conn *connection)`, welche eine Verbindung aus dem Pool entfernt, mehrfach aufgerufen. Innerhalb von `remove` werden zwei Operationen durchgeführt. Zum einen wird die Verbindung aus `p.conns` gelöscht. Dies ist ein assoziativer Speicher, in welchem der `StreamPool` die offenen Verbindungen ablegt. Als zweite Operation wird die Adresse der Verbindung an eine Liste angehangen, in welcher der Speicher alter Verbindungen aufgelistet wird. Aus diesem werden an anderer Stelle neue Verbindungen erzeugt. Wird `remove` mehrfach aufgerufen, werden beide Operationen mehrfach ausgeführt. Das Löschen nicht vorhan-

dener Verbindungen führt zu keinen Fehlermeldungen, wird aber auch nicht durchgeführt. Durch die zweite Operation wird die Verbindung jedoch mehrfach an die Liste der alten Verbindungen angehangen. Bei der Verarbeitung großer Datenmengen wächst diese Liste dadurch sehr stark an, wodurch der Speicherverbrauch massiv wächst. Da nicht bekannt ist, warum die Löschfunktion mehrfach aufgerufen wird, wurde der Speicherfehler dadurch behoben, dass einen zusätzliche Abfrage durchgeführt wird, ob die Verbindung noch in `p.conns` enthalten ist. Nur in diesem Fall wird der Speicher auch an die Liste angehangen.

Verbindungsorientierte Pipeline-Komponenten Auch für die Analyse von Verbindungen existieren die drei Komponenten zum Filtern, Zählen und Ausgeben, welche jedoch im Detail anders funktionieren als die paketorientierten Pendants. Da die beim *PacketFilter* verwendeten *BPF* für Pakete konzipiert sind, lassen sich diese nicht ohne weiteres auf Verbindungen anwenden. Weiterhin haben Verbindungen zusätzliche Parameter, wie etwa Verbindungsdauer oder den Verbindungszustand, welcher ebenfalls zur Filterung berücksichtigt werden kann.
Für die Filterung von Verbindungen kommt daher ein *Scoring* System zum Einsatz. Ein Score ist dabei ein einzelnes Kriterium, auf welches die Verbindung geprüft wird. Mehrere Kriterien können dabei zum selben *Score* beitragen. Zur Filterung wird dann geprüft, ob festgelegte Scores innerhalb eines festgelegten Wertebereichs liegen. Insgesamt kommen fünf Scoring-Elemente zum Einsatz, welche teilweise weitere Konfigurationsmöglichkeiten haben.

Die ersten beiden *Scores* orientieren sich an den Kriterien zum Fingerprinting von Scanning Software, welche von Durumeric [DBH14] für *ZMap* und *Masscan* benannt hat. Beide prüfen für alle eingegangenen Pakete der Verbindung, ob das jeweilige Kriterium des Fingerprints zutrifft. Stimmt dieses bei allen Pakten überein, wird der Score um eins erhöht, ansonsten nicht.
Der *PayloadScore* prüft, ob der Beginn der übertragenen Daten einer Verbindung einem festgelegten Wert entspricht. Dieser Wert wird in der Konfiguration als hexadezimal kodierte Zeichenkette übergeben. Stimmt der Payload mit diesem Wert überein, wird der Score erhöht. Ebenso ist es möglich durch Setzen des Wertes auf eine leere Zeichenkette zu prüfen, ob keine Daten über die Verbindung übertragen wurden.
Durch den *DurationScore* kann geprüft werden, ob die Dauer einer analysierten Verbindung innerhalb festgelegter Grenzen liegt. Dazu kann der Score über einen minimalen und maximalen Wert entsprechend konfiguriert werden. Als Verbindungsdauer wird dabei die Differenz zwischen dem ersten und dem letzten Paket der Verbindung betrachtet.
Die letzte Prüfungsmöglichkeit ist der *HandshakeScore*, welcher den Verbindungsaufbau der Verbindung mit einem Muster vergleicht. Das Muster wird über eine Komma-

getrennte Liste der Werte `1`, `0` und `-1` definiert. Die Liste enthält drei Elemente, wobei jedes einem Paket des Verbindungsaufbaus `SYN, SYN+ACK, ACK` entspricht. Eine `1` gibt dabei an, dass das jeweilige Paket vorhanden sein muss, damit die Prüfung erfolgreich ist, eine `-1`, dass das Paket nicht vorhanden sein darf. Durch eine `0` wird das entsprechende Paket aus der Prüfung herausgenommen. Soll etwa ein Score für die Prüfung auf einen *Stealth Scan* durchgeführt werden, lässt sich dies über die Konfiguration `1,1,-1` abbilden. Das `SYN`, sowie das `SYN+ACK` Paket müssen vorhanden sein, das abschließende `ACK` muss jedoch fehlen, damit es sich um einen *Stealth Scan* handelt.

Um die Scores auf Verbindungen anwenden zu können, existiert die Pipeline-Komponente `Scorer`. Dieser beinhaltet eine Liste von Scores, welche alle auf die Verbindung angewendet werden. Der Scorer speichert die ermittelten Werte im Verbindungsobjekt `TCPStream`. Der *StreamFilter*, welcher die Verbindungen filtert, prüft dann ob die dort hinterlegten Scoring Werte innerhalb der vorgegebenen Parameter liegen.

Der *StreamCounter* funktioniert grundsätzlich wie der *PacketCounter*. Es werden die Anzahl der Verbindungen, der darin enthaltenen Pakete, sowie die Menge der übertragenen Daten aufsummiert, wobei eine getrennte Zählung durch den Einsatz von Klassifikatoren möglich ist. Zusätzlich kann der *StreamCounter* jeweils die für den Filterungsprozess errechneten *Scores* der Verbindungen mitzählen.

Die Ausgabe von zusammengesetzten Verbindungen erfolgt nicht als PCAP-Datei. Dafür gibt es mehrere Gründe. Zum einen müssen dazu bei der Rekonstruktion alle zu dieser Verbindung gehörigen Pakete zwischengespeichert werden. Dies benötigt zusätzlichen Arbeitsspeicher während des an sich schon speicheraufwendigen Verarbeitungsschrittes. Der zweite Grund ist, dass bei einer näheren Betrachtung der Verbindungen diese zunächst erneut zusammengesetzt werden müssen. Stattdessen verwendet der *CsvStreamOutput* zur Ausgabe zusammengesetzter Verbindungen eine CSV-Datei, in der je Zeile die Zusammenfassung einer Verbindung ausgegeben wird. Die ausgegebenen Attribute können dabei in der Konfiguration festgelegt werden, wobei unter anderem die IP-Adresse und TCP-Port von Client und Server, die Anzahl der jeweils übertragenen Pakete sowie der Payload der Verbindung möglich sind. Die Daten können `gzip`-komprimiert generiert werden.

Die Klassifikatoren Die zur Unterteilung von Paketen und Verbindungen in den *Countern* sowie dem *PacketOutput* eingesetzten *Classifier* teilen die zu zählenden Daten für den jeweiligen Prozessschritt auf. Im Gegensatz zu Filtern, welche durch die Unterscheidung von zwei Gruppen den Kontrollfluss innerhalb der Pipeline grob steuern, zerlegen die Klassifikatoren die Daten feingranular innerhalb einer Komponente ohne das dies Auswirkungen auf andere Komponenten hat. Für Pakete und Verbindungen

existiert jeweils ein Interface *PacketClassifier*, bzw. *StreamClassifier*, welche jeweils die Methode `GroupKey(measurement *Measurement) string`, bzw. `GroupKeyStream(s *TCPStream) string` zur Verfügung stellen. Die Methode berechnet aus dem übergebenen Paket oder der übergebenen Verbindung eine Zeichenkette, welcher die Klasse festlegt. Als Klassifikatoren existieren unter Anderem der *PortClassifier*, der *DayClassifier* sowie der *PayloadClassifier*. Der *PortClassifier* gibt als Klasse den Zielport des Aufzeichnungsobjektes zurück, wobei durch die Konfigurationsoption *reverse* auch der Quellport ausgewählt werden kann. Durch den *DayClassifier* wird die Klasse aus dem Zeitpunkt der Aufzeichnung bestimmt. Die Genauigkeit dieses Zeitpunktes wird durch eine Formatierungszeichenkette in der Konfiguration des Klassifikators festgelegt. Im *PayloadClassifier* werden die ersten `n` Bytes der übertragenen Daten als hexadezimale Zeichenkette zurückgegeben. Auch hier wird die genaue Anzahl in der Konfiguration festgelegt. Eine vollständige Auflistung aller Klassifikatoren und Konfigurationsparameter ist in Anhang B angegeben.

Eine vom System automatisch angelegte Komponente ist der *Endpoint*. Dieser wird mit allen Ausgängen von Pipeline-Elementen verknüpft, mit denen kein Eingang anderer Elemente verknüpft ist. Der *Endpoint* nimmt Daten an diesen Ausgängen entgegen und verwirft diese, damit der Ausgangspuffer der vorherigen Elemente nicht überfüllt wird.

5.2. Aufbau der Pipeline

Für die Implementierung der zuvor beschriebenen Pipeline-Architektur ist in Go die Nutzung von zwei verschiedenen Konstrukten möglich. Die erste ist die Nutzung von Go *Channels* als Kommunikationskanal zwischen zwei Pipeline-Komponenten. So werden die verarbeiteten Daten von einer Komponente an die andere weitergereicht. Für Pipeline-Elemente mit mehreren Ausgängen, wie Filter, existieren dann mehrere Channels zur Weitergabe an unterschiedliche Komponenten. Jede Komponente läuft dann in einer eigenen Go Routine, welche aus dem Eingangschannel Daten liest, diese entsprechend ihrer Aufgabe verarbeitet und in den Ausgangschannel schreibt. Die Lese- und Schreiboperationen auf Channels sind jeweils atomar. Um dies zu erreichen verwendet Go Mutexes.

Die zweite Möglichkeit der Implementierung ist die Weitergabe über Funktionsaufrufe. Die aktuelle Komponente ruft dazu die `execute`-Funktion jeder jeweils passenden nächsten auf, sodass die Daten immer als Funktionsparameter weitergereicht werden. Durch den Aufruf als Go-Routine blockiert dieser Aufruf nicht, sodass bereits mit der

Verarbeitung des nächsten Datensatzes fortgefahren werden kann. Diese Methode benötigt im Vergleich zur ersten keine Lock-Mechanismen wie Mutexes, verursacht allerdings zusätzlichen Rechenaufwand durch viele Funktionsaufrufe.

Um zu evaluieren, welches Verfahren eine performantere Verarbeitung der Daten ermöglicht, wird die Laufzeit der Verfahren in Abhängigkeit von Pipeline-Länge und Anzahl der Datensätze gemessen. Dazu werden automatisch Pipelines mit vorgegebener Länge erzeugt und eine vorgegebene Anzahl von Daten durch diese übertragen, jedoch nicht weiter verarbeitet.

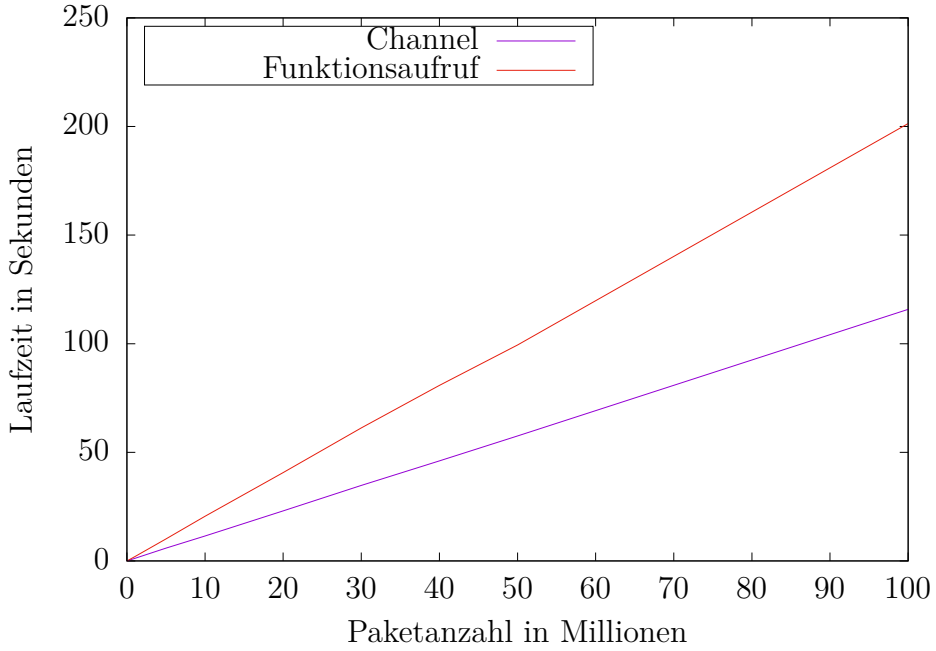


Abbildung 5.1.: Entwicklung der Laufzeit bei 100 Pipeline-Elementen und steigender Anzahl der Pakete

Abbildung 5.1 zeigt die Laufzeit beider Verfahren in Abhängigkeit der Paketanzahl. Die Länge der Pipeline ist dabei fest auf den Wert 100 gesetzt. Beide Verfahren skalieren linear mit der Anzahl der verarbeiteten Pakete, wobei die Implementierung mittels Channel nur etwas mehr als die Hälfte der Laufzeit des anderen Verfahrens benötigt. Weniger deutlich ist die Betrachtung der Laufzeit bei einer festen Paketanzahl und einer steigenden Pipeline-Länge, wie in Abbildung 5.2 dargestellt. Auch hier skalieren beide Verfahren etwa linear mit der Anzahl der hintereinander gereihten Pipeline-Elemente. Im Bereich bis ca. 15000 Pipeline-Elementen ist das Verfahren mittels Funktionsaufruf minimal, jedoch nicht signifikant, schneller.

Für die eigentliche Analyse werden rund 3 Milliarden Pakete analysiert. Die Länge der dazu verwendeten Pipeline wird jedoch in den meisten Fällen deutlich unterhalb von 100

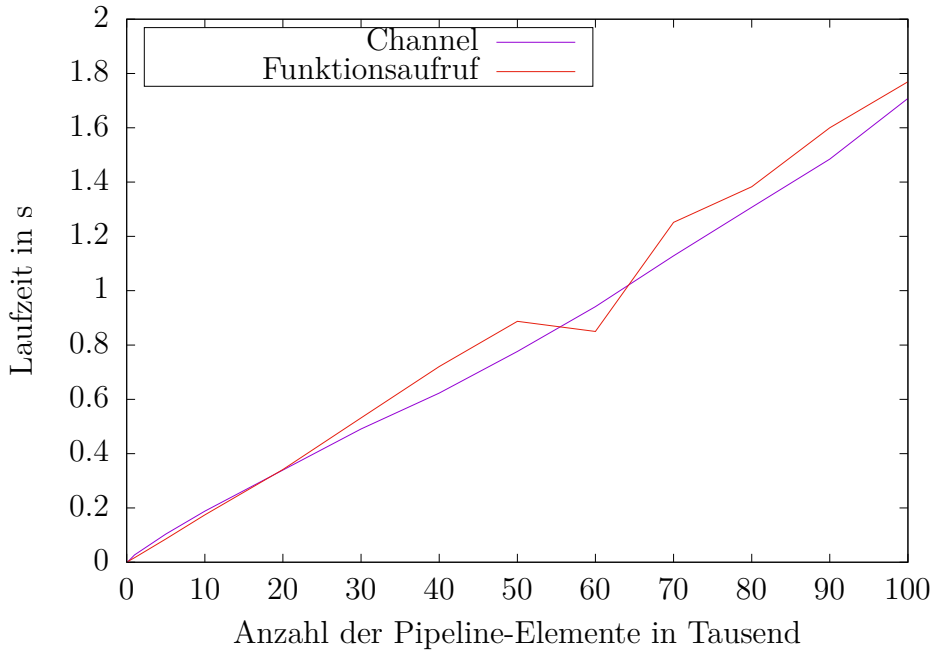


Abbildung 5.2.: Entwicklung der Laufzeit bei 1000 Paketen und steigender Anzahl von Pipeline-Elementen

liegen, sodass vor allem die Betrachtung der ersten Messung für die Analyse relevant ist. In dieser wurde eine deutlich bessere Laufzeit der Channel-Implementierung gemessen, sodass diese für die Implementierung zum Einsatz kommt.

5.2.1. Konfiguration einer Analyse

Die Konfiguration der Analyse erfolgt durch eine XML-Datei, welche die verwendeten Pipeline-Elemente, sowie weitere Eigenschaften definiert. Der Grundaufbau dieser Konfigurationsdatei ist in Abbildung 5.3 dargestellt. Die Konfiguration erfolgt in den vier Abschnitten *Components*, *Connect*, *Settings* und *SpecialPackets*. In Components werden alle verwendeten Pipeline-Elemente definiert. Jedes erhält eine eindeutige Identifikation über das `id`-Attribut. Durch die Verwendung von *Connect*-Elementen werden die Komponenten miteinander verbunden. Im gezeigten Minimalbeispiel wird der *FileWorker*, welcher automatisch erzeugt und durch `FILE` referenziert wird, mit der definierten Parser-Komponente verknüpft. Verbunden werden kann jeweils der Eingang einer Komponente mit einem der Ausgänge einer anderen. Zur Verfügung stehen hierbei `output`, `no` und `other`. Output ist der Standard-Ausgang einer Komponente. Der Ausgang `no` wird von Paketfiltern verwendet, wenn Pakete nicht dem Filterkriterium entsprechen. Der `other` Ausgang wird als zweiter Ausgang von verbindungsorientierten Komponenten

```
<?xml version="1.0" encoding="utf-8" ?>
<AnalysisConfig xmlns="http://vs.uni-due.de/pcap-analyser" >
  <Components>
    <Parser id="parser"/>
  </Components>

  <Connect output="FILE" input="parser"/>

  <Settings>
  </Settings>

  <SpecialPackets>
  </SpecialPackets>
</AnalysisConfig>
```

Abbildung 5.3.: Grundaufbau der Konfigurationsdatei

wie *Composer* oder *StreamFilter* verwendet. Unpassend zusammengefügte Komponenten werden zu Beginn der Analyse erkannt, was zu einem Abbruch der Software mit entsprechender Fehlermeldung führt.

Über *Settings* werden die Rahmenbedingungen der Analyse festgelegt. Dies beinhaltet zu verarbeitenden PCAP-Dateien (`Input`), sowie den Speicherort für Ausgaben der Analyse-Software (`Output`). Weiterhin kann der Name der Log-Datei für die Standardausgabe (`LogFile`), die Größe des verwendeten Lesepuffers für PCAP-Dateien (`BufferSize`) sowie die Anzahl der gleichzeitig verarbeiteten Dateien (`Concurrent`) festgelegt werden. Die Konfiguration der *SpecialPackets* ermöglicht das Ausschließen einzelner aufgezeichneter Pakete aus der Analyse. Dabei wird jeweils der Dateiname, sowie die Position des Pakets innerhalb der Datei angegeben. In Anhang B ist die vollständige Spezifikation des Konfigurationsformats als XML-Schemadatei mit zusätzlichen Ausführungen zur Bedeutung von Elementen und Attributen vorhanden.

Der Aufbau der Pipeline aus der Konfiguration erfolgt automatisch innerhalb der Analysesoftware. Dazu werden zunächst alle Pipeline-Komponenten wie festgelegt erzeugt und in einem assoziativen Speicher anhand ihrer `id` abgelegt. Im Anschluss werden die *Connect* Direktiven ausgewertet. Die jeweils zu verknüpfenden Komponenten werden aus dem Speicher geladen und auf Kompatibilität geprüft. Abschließend werden Ausgänge erkannt, welche mit keinem Eingang verbunden sind. Diese werden mit einem Endpunkt verbunden, welcher die Ausgänge ausließt und die Daten dann verwirft. Dies ist nötig, damit der Puffer der unverbundenen Ausgänge nicht bis zu seiner maximalen Kapazität gefüllt wird, da dies weitere Schreibvorgänge in den Ausgang blockiert. Die erzeugte Pipeline-Struktur wird entsprechend der in `Concurrent` festgelegten Anzahl von Workern kopiert, sodass jeder Worker eine eigene unabhängige Verarbeitungspipeline hat und keine *Race Conditions* entstehen.

5.2.2. Fehlerbehandlung

Während der Analyse der PCAP-Dateien können verschiedenen Fehler auftreten, welche Einfluss auf die Ergebnisse der Analyse nehmen können. Einer dieser Fehler sind unvollständige Pakete am Ende der aufgezeichneten PCAP-Dateien. Vermutlich sind diese durch Absturz der Aufzeichnungssoftware entstanden, wenn Daten im Schreibpuffer nicht mehr ordnungsgemäß auf die Festplatte geschrieben wurden. Da dies nur beim letzten Paket einer Datei auftreten kann, ist die Anzahl dieser Fehler auf die Anzahl der gespeicherten Dateien, also 25, begrenzt. Tatsächlich sind fünf Pakete davon betroffen. Die betroffenen Pakete werden bei Auslesen der PCAP-Dateien erkannt und verworfen. Beim Umschreiben der PCAP-Dateien werden diese ebenfalls ignoriert und nicht in die neuen Dateien geschrieben.

Für vier TCP-Pakete ist durch Abstürze in der Implementierungsphase bekannt, dass diese die zur Rekonstruktion der TCP-Pakete verwendete Bibliothek `gopacket` in eine Endlosschleife versetzten. Die genaue Fehlerursache ist jedoch unbekannt, sodass zur Fehlervermeidung die bekannten Pakete grundsätzlich von der Analyse ausgeschlossen werden. Aufgrund der geringen Anzahl dieser Pakete hat dies keinen signifikanten Einfluss auf das Ergebnis der Analyse.

Ein ebenfalls möglicher Fehler sind inkorrekte Daten innerhalb des aufgezeichneten Datenverkehrs. Dies beinhaltet sowohl Übertragungsfehler, als auch absichtlich gefälschte Daten. Möglich ist etwa, dass der IP-Header als nächstes Protokoll TCP benennt, jedoch keinen korrekten TCP-Header enthält. Die zur Dekodierung der Pakete verwendete Bibliothek `gopacket` bietet Funktionalität zur Überprüfung der Protokolle an, welche in den Pipeline-Komponenten zum Einsatz kommt. Fehler werden in einem gesonderten Fehler-Log protokolliert.

5.3. Optimierungen und Evaluation

Die Analyse erfolgt auf einer virtuellen Maschine (VM) auf Basis von KVM-Virtualisierung. Für die VM sind 12 Prozessorkerne mit je 3 GHz Taktung, 32 GB Arbeitsspeicher und ausreichend Festplattenspeicher auf Basis eines RAID5 verfügbar. Zur Einordnung der Lesegeschwindigkeit wurde mittels `md5sum` eine Prüfsumme für eine große PCAP-Datei gebildet, welche etwa 49 GB groß ist. Diese wurde innerhalb von 2 Minuten und 1 Sekunden verarbeitet, was einem Datendurchsatz von 411 MB/s entspricht. Dieser Wert bietet eine Orientierung für die mögliche Lesegeschwindigkeit der Daten.

5.3.1. Effiziente Eingabeverarbeitung

Die Verarbeitung von PCAP-Dateien erfordert grundsätzlich viele Leseoperationen auf der Festplatte. Es kann jeweils der 16 Byte große *Record Header* eines Pakets ausgelesen werden, aus welchem die Größe des eigentlichen Pakets hervorgeht. Dieses wird dann in einem zweiten Lesevorgang von der Festplatte geladen. Das Auslesen vieler kleiner Datenfragmente ist auf rotierenden Speichermedien wie Festplatten jedoch zeitaufwändig, da bei jedem Lesevorgang der Lesekopf bewegt und gewartet werden muss, bis die rotierende Platte zur richtigen Stelle gedreht ist. Diese Wartezeiten summieren sich bei vielen Operationen auf.
Dieses Problem kann jedoch durch den Einsatz eines Pufferspeichers behoben werden. Bei Leseoperationen auf der Festplatte werden direkt mehrere Megabyte Daten ausgelesen und im Arbeitsspeicher zwischengespeichert. Das Auslesen der PCAP-Daten erfolgt dann direkt aus dem Arbeitsspeicher, welcher deutlich schnelleren Zugriff bei vielen Anfragen nach kleinen Datenmengen ermöglicht.

5.3.2. Parallelisierung

Um eine möglichst gleichmäßige Auslastung der Analyse-VM zu erreichen ist eine parallele Verarbeitung mehrerer PCAP-Dateien sinnvoll. Durch die zuvor beschriebene Verwendung eines Puffers für Lesezugriffe auf die Festplatte, wird diese für die Zeit, in der die Daten im Puffer verarbeitet werden, nicht genutzt. In dieser Zeit kann jedoch bereits der Lesevorgang für den Puffer einer anderen PCAP-Datei stattfinden. Die Analyse erfolgt daher zumeist für mehrere PCAP-Dateien gleichzeitig. Für jede parallele Verarbeitung existiert ein *Worker Thread*, welcher diese in einer eigenen Pipeline durchführt. Dies hat den Vorteil, dass mehrere Worker keine gemeinsam genutzten Speicherbereiche, etwa zum Zählen von Paketen, haben. Bei gleichzeitigem Zugriff auf dieselben Speicherbereiche müsste sonst zur Vermeidung von *Race Conditions* durch die Verwendung von *Locks* ein atomarer Zugriff sichergestellt werden. Dies ist jedoch bei hoher Parallelisierung ein hoher zusätzlicher Aufwand. Die getrennten Speicherbereiche werden nach Abschluss der Analyse zusammengeführt und dann als Bericht auf die Festplatte geschrieben.

Eine Parallelisierung ist jedoch nicht immer möglich. Wie in Abschnitt 4.3 beschrieben gibt es für die TCP-Rekonstruktion Grenzen der Parallelisierbarkeit, wenn Segmente der selben Verbindung über mehrere Dateien verteilt sind. Um dies zu umgehen ist es nötig die PCAP-Dateien so umzuschreiben, dass alle zu einer Verbindung gehörenden Pakete in der gleichen Datei sind. Diese Umschreibeoperation selbst ist ebenfalls nicht parallelisierbar. Diese erfolgt durch eine sequentielle Verarbeitung aller PCAP-Dateien in chronologischer Reihenfolge. So wird sichergestellt, dass jede Zieldatei die Pakete

ebenfalls in chronologischer Reihenfolge enthält. Dies ist für die Rekonstruktion der TCP-Verbindungen mittels Sliding-Window wichtig.

Die Verarbeitungsreihenfolge bei paralleler Verarbeitung mehrerer Dateien erfolgt nach dem Kriterium der Größe. Große Dateien werden zuerst verarbeitet, kleinere später. Dadurch soll verhindert werden, dass viele *Worker* bereits beendet sind, da keine weiteren Dateien zur Verarbeitung mehr vorhanden sind, während wenige Worker noch die Verarbeitung von großen Dateien durchführen. Da die Größe der Dateien auch nach dem Umschreibeprozess noch stark voneinander abweicht, wird durch dieses Verfahren eine gleichmäßigere Auslastung ermöglicht.

5.3.3. Performance-Evaluation

Um die Wirksamkeit der beschriebenen Maßnahmen zur Verbesserung der Performance zu zeigen und zu quantifizieren, wird eine Messung der Laufzeit unter verschiedenen Bedingungen vorgenommen. Für diese Messung werden acht etwa 600 MB große PCAP-Dateien des Datensatzes verwendet. Da der verwendete Linux-Kernel einen eigenen Cache für Dateisystemzugriffe verwendet, kann die Messung verfälscht werden. Werden mehrere Messungen durchgeführt, können die Daten ab dem zweiten Durchlauf aus dem Cache geladen werden, ohne dass ein Zugriff auf die Festplatte notwendig ist. Um dies bei der Evaluation auszuschließen wird vor jeder Messung der Cache durch den Befehl `echo 3 > /proc/sys/vm/drop_caches`[7] geleert.

Zunächst wird verglichen, welchen Einfluss die Wahl der Bibliothek zum Datei-Einlesen auf die Laufzeit der Analyse hat. Für `libpcap` und die Go-Implementierung `pcapgo` unter Einsatz eines Puffers ist die Laufzeit in Abhängigkeit des verwendeten Parallelisierungsgrades in Abbildung 5.4 dargestellt. Die Verwendung von `pcapgo` mit Puffer ist insbesondere bei einem Parallelisierungsgrad von 8 etwa 65% schneller als die `libpcap` Implementierung.

Welche Auswirkungen die Puffergröße auf die Laufzeit der Software hat ist in Abbildung 5.5 dargestellt. Eine Reduktion der Laufzeit ist bis zu einer Puffergröße von 16 MB deutlich erkennbar. Bei weiterer Steigerung bis 128 MB wird die Laufzeit nur noch minimal reduziert.

Zur Feststellung der Laufzeit der Analyse für verschiedene Anwendungsfälle wird eine exemplarische PCAP-Datei durch verschiedene Konfigurationen der entwickelten Analysesoftware verarbeitet und die Laufzeit der Ausführung gemessen. Dazu werden fünf Szenarien konfiguriert (Tabelle 5.1). Die ersten drei Szenarien beschränken sich auf die

[7] `http://linuxwiki.de/proc/sys/vm#A.2Fproc.2Fsys.2Fvm.2Fdrop_caches`

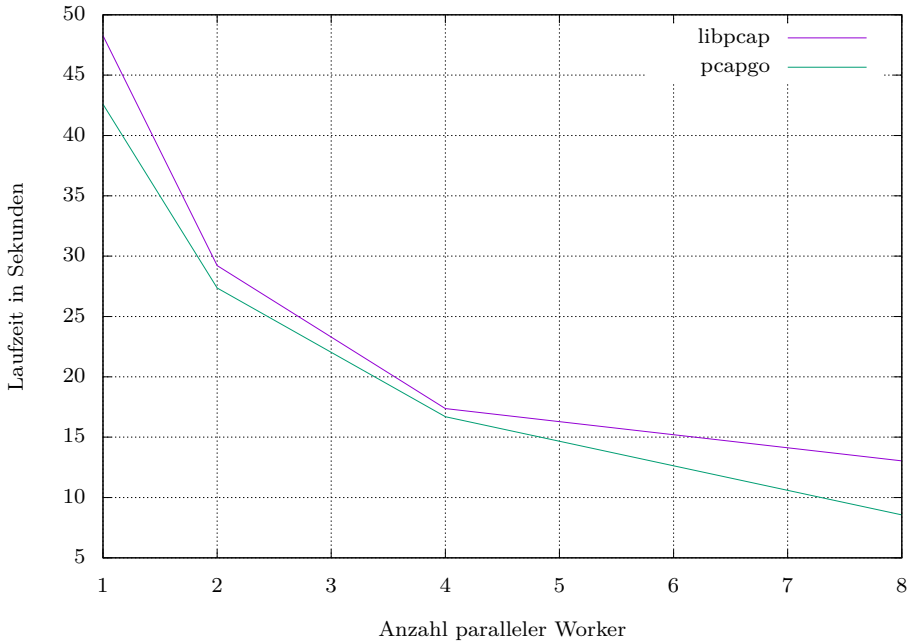


Abbildung 5.4.: Laufzeit für das Einlesen des Testdatensatzes in Abhängigkeit der Anzahl parallel verarbeiteter Dateien

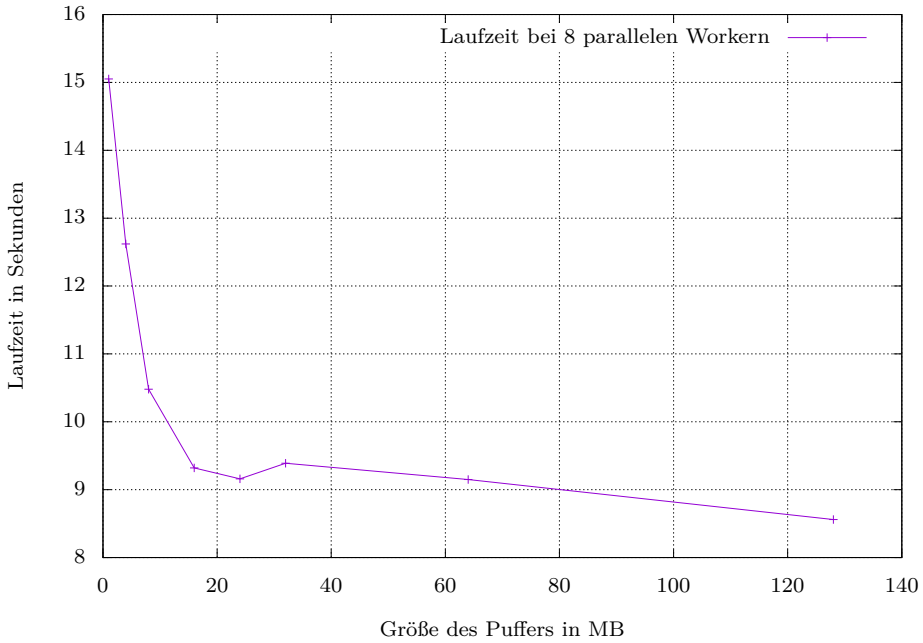


Abbildung 5.5.: Laufzeit für das Einlesen des Testdatensatzes in Abhängigkeit der Puffergröße

Verarbeitung von Paketen. Beim ersten werden lediglich die Daten eingelesen und durch die *Parser* Komponente verarbeitet. Im zweiten Szenario wird zusätzlich eine Zählung der Pakete vorgenommen. Eine vor der Zählung vorgenommene Filterung des eingehenden Datenverkehrs findet im dritten Szenario statt.
Die Szenarien vier und fünf verwenden den *Composer* zur Zusammensetzung von TCP-Verbindungen und zählen diese. Zusätzlich dazu wird im fünften ebenfalls eine Filterung der TCP-Verbindungen durch das zuvor beschriebene Scoring System vorgenommen.

Die Durchführung des ersten Szenarios hat eine Laufzeit von 33 Sekunden. Dies entspricht einem Datendurchsatz von etwa 500 MB/s. Durch die zusätzliche Verwendung eines Zählers verlängert sich die Laufzeit auf 128 Sekunden, bei vorheriger Filterung jedoch nur auf 54 Sekunden. Ein Grund hierfür ist der im *Lazy*-Modus arbeitende *Parser*. Das bedeutet, dass das Parsen einzelner Protokollschichten erst dann vorgenommen wird, wenn diese benötigt werden. Wird also auf die Paketdaten nicht zugegriffen, wie es im ersten Szenario der Fall ist, entfällt der Vorgang fast vollständig. Dieser wirkt sich erst im zweiten Szenario auf die Laufzeit aus, da für die zusätzliche Zählung das Parsen notwendig ist. Die Filterung selbst arbeitet durch den Einsatz von BPF auf den ungeparsten Bytes, sodass der Filterprozess die Anzahl der Parser-Operationen reduziert und zu einer Verringerung der Laufzeit führt.

Bei Durchführung der TCP-Rekonstruktion erhöht sich die Laufzeit signifikant. Szenario 4, welches TCP-Verbindungen zusammensetzt und zählt, benötigt eine Laufzeit von 222 Sekunden. Durch zusätzliche Filterung in Szenario 5 verringert sich die Laufzeit minimal auf 207 Sekunden. Die Filterung von Verbindungen ist aufgrund der Payload-Analyse rechenintensiver als die einfache Filterung auf Paketebene. Eine Übersicht der Laufzeiten und des Datendurchsatzes ist in Tabelle 5.1 dargestellt. Zusammengefasst zeigt diese Auswertung, dass die paketbasierte Verarbeitung bei passender Filterung der relevanten Daten von `pcap-analyser` mit einem hohen Datendurchsatz durchgeführt werden kann, wohingegen die Zusammensetzung der TCP-Verbindungen eine zeit- und rechenintensive Operation ist.

Szenario	**Laufzeit (s)**	**Durchsatz (MB/s)**
lesen; parsen	32	499,07
lesen; parsen; zählen	128	124,77
lesen; parsen; filtern; zählen	54	295,75
lesen; parsen; tcp; zählen	222	71,94
lesen; parsen; tcp; filtern; zählen	207	77,15

Tabelle 5.1.: Laufzeit und Datendurchsatz der Testszenarien

5.3.4. Evaluation der TCP-Rekonstruktion

Um die Funktion der TCP-Rekonstruktion zu evaluieren, wurde ein unabhängiger Testdatensatz[8] mit Netzwerkaufzeichnungen im PCAP-Dateiformat verwendet. Dieser wurde durch die etablierte Software `Wireshark`, sowie die im Rahmen dieser Arbeit konzipierte Analysesoftware (`pcap-analyser`) verarbeitet, wobei jeweils die Anzahl der TCP-Verbindungen ermittelt wurde.
Bei der Analyse der Datei `smallFlows.pcap` ermittelte `Wireshark` 484 TCP-Verbindungen, die eigene Analysesoftware hingegen 391. Für den zweiten Datensatz `bigFlows.pcap` beträgt die Anzahl der gezählten Verbindungen 22.329, bzw. 19.919. Die Unterschiede beruhen darauf, dass jeweils unterschiedliche Kriterien für eine Verbindung genutzt werden.

`Wireshark` zählt auch Verbindungssegmente ohne zugehöriges `SYN`-Flag als Verbindung, sodass hier gezählte Verbindungen teilweise nur aus einem Verbindungsabbau bestehen. Im Gegensatz dazu fügt `pcap-analyser` nur Verbindungen zusammen, für welche zumindest in eine Richtung ein `SYN`-Flag vorliegt. Nicht zugeordnete Segmente werden vom *Composer* gesondert markiert, sodass diese von der Zählung explizit ausgenommen werden können. Durch Anpassung der Filtereinstellungen in `Wireshark` auf die Bedingung, dass zumindest ein `SYN`-Flag in jeder Verbindung vorhanden sein muss, stimmen die Zahlen für beide Dateien exakt überein. Daher ist davon auszugehen, dass `pcap-analyser` die Verbindungen korrekt zusammensetzt. Da TCP zwingend einen Verbindungsaufbau voraussetzt, bildet eine Filterung, wie sie von `pcap-analyser` vorgenommen wird, die Realität präziser ab.

[8] `http://tcpreplay.appneta.com/wiki/captures.html`

6. Auswertung

Der vorliegende Datensatz umfasst Aufzeichnung des Netzwerkverkehrs von 95 IP-Adressen, auf welchen über einen Zeitraum vom 16.11.2012 bis zum 23.12.2016 eine semi-aktive Messung durchgeführt wurde. Innerhalb des gesamten Messzeitraums von 1498 Tagen wurden über einen Zeitraum von 290 Tagen aufgrund technischer Fehler keine Daten gespeichert. Somit liegen für 1208 Tage Daten vor; dies entspricht 80,6% des gesamten Messzeitraums. Die Mitschnitte liegen in 25 Dateien `bh-wan01.pcap` bis `bh-wan25.pcap` vor, welche jeweils die Messung bis zu einer Unterbrechung des Mitschnitts beinhalten. Die genauen Zeiträume, welche im Mitschnitt erfasst sind, stellt Tabelle 6.1 dar.

Datei	Erstes Paket	Letztes Paket	Tage	eing. Pakete	ausg. Pakete
bh-wan01.pcap	2012-11-16 16:57:26	2012-11-26 22:18:45	10,2	2.082.686	1.328.822
bh-wan02.pcap	2012-11-26 22:19:01	2013-01-04 18:33:21	38,8	120.765.981	72.061.881
bh-wan03.pcap	2013-01-04 18:33:24	2013-01-08 12:37:06	3,8	86.887.092	51.427.074
bh-wan04.pcap	2013-01-08 20:55:54	2013-01-14 12:59:15	5,7	33.726.236	19.929.534
bh-wan05.pcap	2013-01-14 12:59:22	2013-03-11 15:03:17	56,1	327.383.373	194.455.288
bh-wan06.pcap	2013-03-11 15:03:19	2013-05-27 12:44:23	76,9	363.003.619	219.054.697
bh-wan07.pcap	2013-05-27 12:44:26	2013-06-12 13:47:10	16,0	216.265.481	128.936.739
bh-wan08.pcap	2013-06-12 15:26:29	2013-07-11 13:22:04	28,9	78.492.251	47.511.201
bh-wan09.pcap	2013-07-11 13:22:12	2013-09-05 11:24:05	55,9	17.583.818	10.948.059
bh-wan10.pcap	2013-09-05 11:24:13	2014-03-06 17:20:36	182,2	102.133.391	54.474.233
bh-wan11.pcap	2014-04-01 17:36:58	2014-05-27 14:09:02	55,9	6.815.261	4.757.518
bh-wan12.pcap	2014-07-02 11:11:53	2014-07-29 14:26:16	27,1	3.898.235	2.553.574
bh-wan13.pcap	2014-07-29 14:52:57	2014-08-20 08:23:43	21,7	4.011.888	2.609.904
bh-wan14.pcap	2014-10-27 10:31:38	2014-10-31 07:38:32	3,9	497.460	354.816
bh-wan15.pcap	2014-11-29 18:19:32	2015-01-12 07:22:01	43,5	11.229.612	7.957.388
bh-wan16.pcap	2015-02-04 14:12:35	2015-04-28 08:58:36	82,8	27.111.675	18.846.774
bh-wan17.pcap	2015-05-01 01:18:41	2015-05-06 21:38:29	5,8	1.652.911	1.099.780
bh-wan18.pcap	2015-05-06 21:40:08	2015-09-01 03:14:41	117,2	48.849.681	33.963.114
bh-wan19.pcap	2015-10-06 13:59:05	2015-11-23 07:51:10	47,7	19.685.625	10.564.159
bh-wan20.pcap	2015-12-22 00:31:07	2016-04-11 08:10:51	111,3	20.563.068	14.714.726
bh-wan21.pcap	2016-05-12 22:52:00	2016-06-20 07:20:00	38,4	16.961.577	15.360.340
bh-wan22.pcap	2016-06-28 09:57:45	2016-08-08 07:36:10	40,9	30.133.082	24.154.845
bh-wan23.pcap	2016-08-08 08:25:52	2016-10-18 14:34:35	71,3	69.466.338	58.881.950
bh-wan24.pcap	2016-10-18 16:08:24	2016-10-31 08:04:47	12,7	20.177.003	17.173.091
bh-wan25.pcap	2016-10-31 11:47:03	2016-12-23 17:04:22	53,2	120.172.212	108.728.235

Tabelle 6.1.: Zeiträume und Datenumfang der vorliegenden PCAP-Dateien

In der Aufzeichnung befinden sich 1.749.549.554 eingehende und 1.121.847.739 ausgehende Ethernet-Pakete. Über den gesamten Messzeitraum wurden insgesamt 129,7 GB eingehende und 64,8 GB ausgehende Daten übertragen und aufgezeichnet. Alle Ethernet-Pakete enthalten IPv4 Datenverkehr. Der Ursprung des eingegangenen Datenverkehrs ist

in Abbildung C.1 als Heatmap dargestellt. An dieser Darstellung ist erkennbar, dass Pakete aus verschiedenen Adressbereichen des Internets empfangen wurden. Adressbereiche, aus denen keine Daten empfangen wurden, sind hauptsächlich reservierte Adressen und die in den Anfangszeiten des Internet vergebenen Subnetze an große Unternehmen. Es wurden beispielsweise keine Daten aus dem Netz 9.0.0.0/8 empfangen, welches IBM zugewiesen ist. In der feingranulareren Darstellung in Abbildung C.2 ist erkennbar, dass aus dem Adressbereich von 64.0.0.0 bis 128.0.0.0 sowie im Bereich von 176.0.0.0 bis 224.0.0.0 mehr Subnetze Pakete gesendet haben als aus den übrigen Adressbereichen.

Der Anteil der jeweils in den IPv4-Pakten übertragenen Protokolle wird in Tabelle 6.2 dargestellt. Die Angabe der Bytes bezieht sich für TCP, UDP und ICMP auf den im jeweiligen Protokoll übertragenen Payload. Für Andere ist der Payload des IP-Pakets angegeben. Wie bei der vorherigen Analyse des Datensatzes im Jahr 2013 [Sch13] ist TCP mit rund 98,8% der Pakete und 93,8% des Datenvolumens das am meisten genutzte Protokoll. Der UDP-Datenverkehr wird in etwa 1% der Pakete genutzt, ist jedoch überproportional für das eingehende Datenvolumen verantwortlich. Die Ursache dafür liegt zumeist daran, dass in TCP mehr Pakete ohne Payload versendet werden, welche für die Verwaltung der Verbindung, etwa dessen Auf- und Abbau, zuständig sind. Der Anteil von ICMP-Pakten liegt bei etwa 0,1%.

Protokoll	**Pakete**	**% Pakete**	**Bytes**	**% Bytes**
TCP	1.729.217.235	98,8	30.415.104.321	93,8
UDP	17.932.402	1,0	1.916.457.280	5,9
ICMP	2.255.909	0,1	94.235.149	0,3
Andere	1030	0,0	71.388	0,0

Tabelle 6.2.: Anteile der im Paket befindlichen Protokolle

Die anderen Protokolle bestehen zu über 80% aus SCTP. Hierbei handelt es sich um ein Transportprotokoll, welches wie TCP eine Verbindungsorientierung auf einer paketvermittelten Kommunikation etablieren soll, jedoch zusätzliche Schutzmaßnahmen gegen Denial-of-Service Angriffe implementiert. Es ist daher zu vermuten, dass es sich beim eingehenden SCTP-Traffic um Scans handelt. Die übrigen Pakete lassen sich aufgrund des `protocol`-Feldes im IP-Header teilweise Tunnel-Protokollen zuordnen. Es wurden aber auch Werte festgestellt, die keinem Protokoll zugeordnet sind. Das Gesamtvolumen dieser Pakete ist vernachlässigbar.

Im Folgenden werden die Protokolle ICMP, UDP und TCP einzeln betrachtet und wie in Abschnitt 4.1 beschrieben klassifiziert.

6.1. ICMP

Die Auswertung der eingegangenen ICMP-Pakete (Tabelle 6.3) erfolgt nach dem zuvor beschriebenen Klassifizierungsverfahren anhand des ICMP-Headers, sowie der Prüfung, ob der Payload ein TCP-Paket enthält, welches von der aktiven Antwortkomponente des Messnetzwerks versendet wurde (Tabelle 4.1).

Klassifizierung	Pakete	% Pakete	Bytes	% Bytes
Artefakte	1.088.096	48,2	46.996.101	49,9
Scan	748.476	33,2	12.776.690	13,6
Backscatter	419.337	18,6	34.462.358	36,6

Tabelle 6.3.: Klassifizierung der empfangenen ICMP-Pakete

Bei etwa der Hälfte der Pakete (48,2%) enthält der Payload ein solches TCP-Paket. Es ist daher davon auszugehen, dass durch den Einsatz des *active responders* doppelt so viele ICMP-Pakete empfangen werden, als es bei einer Messung mittels eines vollständig passiven Darknets der Fall ist. Das aktive Beantworten des TCP-Datenverkehrs wirkt sich also auch auf den Umfang des gemessen ICMP-Datenverkehrs aus, auch wenn dieser nur passiv gemessen und nicht beantwortet wird.

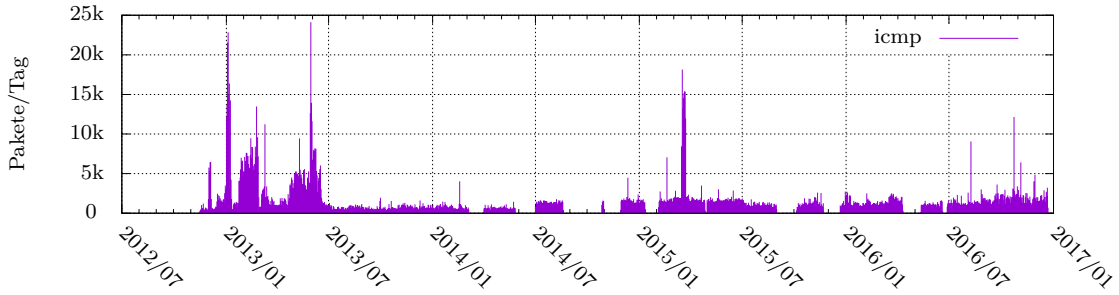


Abbildung 6.1.: Zeitliche Entwicklung der eingegangenen ICMP-Pakete

Ein Drittel der ICMP-Pakete sind Echo Requests, welche genutzt werden um zu prüfen, ob eine IP-Adresse von einem aktiven Host genutzt wird. Daher werden diese als Scan klassifiziert. Das übrige Sechstel sind zum Großteil Pakete des Typs 3 *Destination Unreachable*, welche kein TCP-Paket der aktiven Antwortkomponente enthalten. Diese sind als Backscatter anzusehen.
Der ICMP-Datenverkehr beinhaltet demnach etwa doppelt so viel Scanning wie Backscatter.

6.2. UDP

Einen ersten Überblick zur Analyse des empfangenen UDP-Datenverkehrs lässt sich der Auflistung der zehn häufigsten Zielports in Tabelle 6.4 entnehmen. Die Aufschlüsselung erfolgt dabei sowohl nach Anzahl der Pakete als auch nach Summe der übertragenen Bytes. 55,9% der Pakete und 66,4% der Daten wurden dabei jeweils an diese zehn Ports gesendet. Besonders markant ist, dass auf Port 5060 fast ein Drittel des UDP-Payloads empfangen wird. Dieser Port wird vom *Session Initiasation Protocol (SIP)* genutzt, welches beim Verbindungsaufbau von *Voice-over-IP (VoIP)* Telefonaten zur Anwendung kommt. Hier soll eine Auswertung des Payloads die Frage klären mit welcher Intention diese Pakete versendet wurden. Auffällig ist weiterhin, dass auf Port 8000 zwar nur ein Promille der UDP-Pakete empfangen wird, diese jedoch über 10% des Datenvolumens verursachen. Auch hier soll die Ursache durch eine genauere Betrachtung des Payloads geklärt werden.

10 häufigste Ports nach Paketanzahl					10 häufigste Ports nach Payload				
Port	**Pakete**	**%**	**Bytes**	**%**	**Port**	**Pakete**	**%**	**Bytes**	**%**
20167	2.727.394	15,2	193.646.855	10,1	5060	1.401.609	7,8	592.972.780	30,9
53413	2.358.240	13,2	110.893.475	5,8	8000	17.366	0,1	228.252.381	11,9
8611	1.527.586	8,5	24.441.376	1,3	20167	2.727.394	15,2	193.646.855	10,1
5060	1.401.609	7,8	592.972.780	30,9	53413	2.358.240	13,2	110.893.475	5,8
20799	431.477	2,4	21.850.271	1,1	20160	348.192	1,9	36.559.818	1,9
20160	348.192	1,9	36.559.818	1,9	60447	184.568	1,0	27.685.096	1,4
48457	338.224	1,9	18.593.861	1,0	8611	1.527.586	8,5	24.441.376	1,3
5402	297.738	1,7	12.024.836	0,6	20799	431.477	2,4	21.850.271	1,1
20250	297.165	1,7	10.086.644	0,5	48457	338.224	1,9	18.593.861	1,0
59418	296.880	1,7	11.937.216	0,6	55270	169.379	0,9	18.462.194	1,0
Summe	17.932.402	55,9	1.916.457.280	53,9	Summe	17.932.402	53,0	1.916.457.280	66,4

Tabelle 6.4.: 10 häufigste UDP-Zielports aufgeschlüsselt nach Paketanzahl und Payload

Unter den Top 10 Zielports sind weiterhin einige, welche sich nicht klar einem Protokoll zuordnen lassen. Ursache hierfür kann Backscatter Traffic sein. Diese These wird durch die Betrachtung der zehn häufigsten Quellports (Tabelle 6.5) gestützt. Der mit Abstand häufigste Quellport ist bei rund einem Viertel (23,4%) der Pakete 53, welcher dem *Domain Name System (DNS)* zuzuordnen ist. Hier soll untersucht werden, ob sich einzelne Denial of Service Angriffe auf DNS-Server identifizieren lassen. Ebenso fällt auf, dass die Quellports 39868 und 39642 zusammen in Paketanzahl und Datenvolumen etwa dem Zielport 8000 entsprechen.

Eine Gruppierung der UDP-Pakete nach den ersten 32 Byte des Payloads zeigt vor allem das Vorhandensein von DNS-Antworten auf, was sich mit der Betrachtung der Quellports deckt. Auffällig sind weiterhin Daten der Form `AAAAAA cd /tmp || cd /var/ || ? .`

Diese deuten auf den Versuch hin eine bekannte Schwachstelle mittels eines Exploits auszunutzen, bei welchem ein Shellbefehl auf dem Zielsystem ausgeführt wird.

10 häufigste Ports nach Paketanzahl					10 häufigste Ports nach Payload				
Port	**Pakete**	**%**	**Bytes**	**%**	**Port**	**Pakete**	**%**	**Bytes**	**%**
53	4.198.328	23,4	341.969.495	17,8	53	4.198.328	23,4	341.969.495	17,8
35691	257.470	1,4	9.054.982	0,5	39868	8961	0,0	127.445.971	6,7
60729	215.910	1,2	4.957.832	0,3	39642	7131	0,0	100.175.659	5,2
56731	63.531	0,4	2.208.300	0,1	5070	43.703	0,2	28.567.564	1,5
61345	62.572	0,3	2.188.038	0,1	5071	35.293	0,2	21.760.413	1,1
61445	52.826	0,3	1.796.009	0,1	5061	34.009	0,2	14.064.893	0,7
1024	46.337	0,3	1.931.150	0,1	5062	32.402	0,2	13.435.034	0,7
51413	44.902	0,3	1.812.566	0,1	5074	23.549	0,1	13.053.469	0,7
5070	43.703	0,2	28.567.564	1,5	5060	28.682	0,2	11.997.609	0,6
52260	40.390	0,2	1.418.322	0,1	5064	24.094	0,1	9.935.472	0,5
Summe	17.932.402	28,0	1.916.457.280	20,7	Summe	17.932.402	24,7	1.916.457.280	35,6

Tabelle 6.5.: 10 häufigste UDP-Quellports aufgeschlüsselt nach Paketanzahl und Payload

Durch die Auswertung der häufigsten Ports und Nutzdatenpräfixe wurden die Hauptursachen für den auftretenden UDP-Datenverkehr identifiziert. Diese werden im Folgenden jeweils genauer betrachtet.

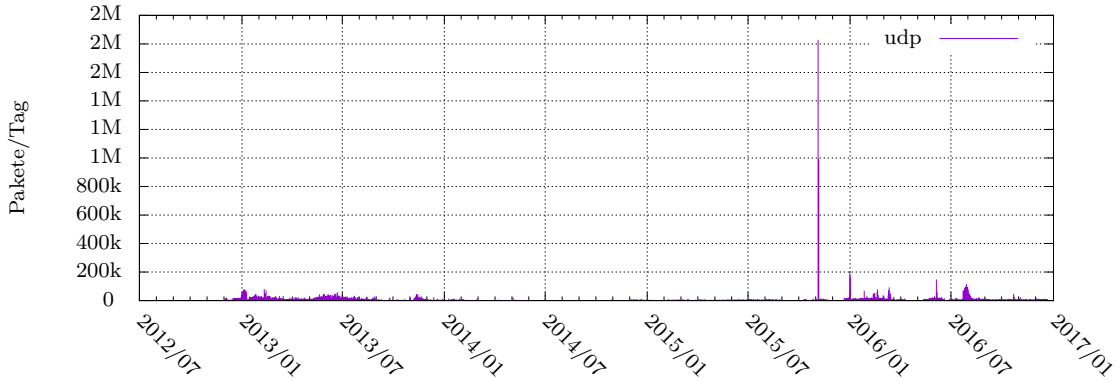


Abbildung 6.2.: Zeitliche Entwicklung der eingegangenen UDP-Pakete

6.2.1. DNS-Antworten

Beim eingegangenen UDP-Datenverkehr mit Zielport 53 handelt es sich zu 99,87% um DNS-Antwortpakete, welche zu 99,75% als Erwiderung auf die Anfrage eines *A Records*, also der Auflösung eines Namens zu einer IP-Adresse gestellt wurden. Dies entspricht den Charakteristiken von Backscatter Datenverkehr, welcher durch Denial-of-Service Angriffe gegen Nameserver entsteht.

65% dessen lässt sich auf einen Einzelnen Angriff auf einen Nameserver innerhalb der von Google betriebenen Cloud-Dienste (`ns-cloud-d4.googledomains.com`) am 05.11.2015 zurückführen. Dieser wurde über einen Zeitraum von mehr als 24 Stunden mit einer stabilen Paketrate von etwa 4000 Paketen pro Minute über eine gefälschte Adresse des Messnetzwerks mit Anfragen des A Records von `www.370sf.com` beschäftigt. Dieses Ereignis ist innerhalb des UDP-Datenverkehrs des Darknets im zeitlichen Verlauf klar erkennbar (Abbildung 6.2).

Über 93% des DNS-Backscatter sind durch Angriffe auf die zehn häufigsten Ziele, welche in Tabelle 6.6 aufgelistet sind, entstanden. Neben dem Angriff auf den Google Nameserver gab es mehrere auf Ziele in China. Da das Messnetzwerk mit 95 Adressen sehr klein ist, können keine sicheren Hochrechnungen über das gesamte Ausmaß dieser Angriffe gemacht werden.

IP-Adresse	Pakete	Land	AS
216.239.38.109	2.727.347	United States	AS15169 Google Inc.
218.60.112.225	348.158	China	AS4837 CNCGROUP China169 Backbone
58.220.46.194	184.566	China	AS23650 AS Number for CHINANET jiangsu province
59.56.73.2	169.378	China	AS133774 Fuzhou
140.205.228.23	124.157	China	AS37963 Hangzhou Alibaba Advertising Co. Ltd.
42.120.221.11	123.495	China	AS37963 Hangzhou Alibaba Advertising Co. Ltd.
218.60.112.224	74.245	China	AS4837 CNCGROUP China169 Backbone
115.236.151.191	65.029	China	AS58461 No.288 Fu-chun Road
113.5.81.69	59.543	China	AS4837 CNCGROUP China169 Backbone
107.191.99.216	52.778	United States	AS3842 RamNode LLC

Tabelle 6.6.: 10 häufigsten Ziele von DNS Denial-of-Service Angriffen

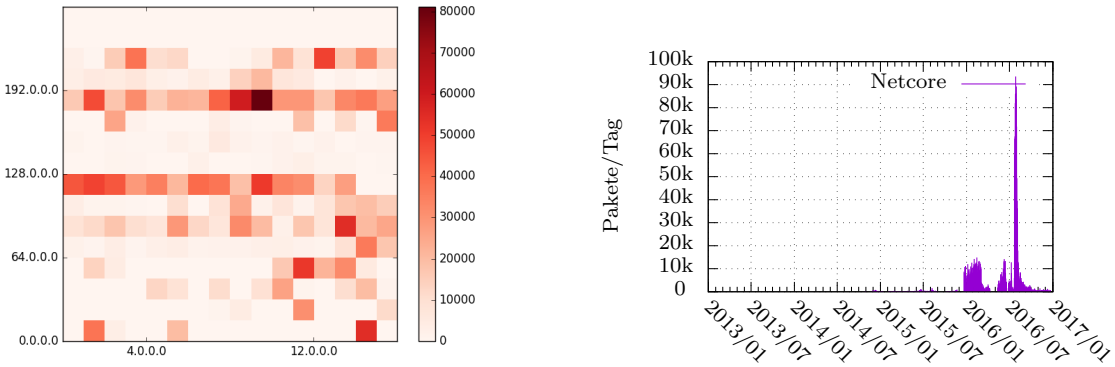
6.2.2. Netcore

Der Zielport UDP 53413 ist für einen Exploit in einem Router der Firma Netcore bekannt[1], welcher im Januar 2016 öffentlich wurde[2]. Etwa ab diesem Zeitpunkt kann auch im aufgezeichneten Datenverkehr ein Anstieg von Paketen mit diesem Zielport festgestellt werden (Abbildung 6.3b). Die Anzahl der eingehenden Pakete nimmt daraufhin kurzzeitig ab, steigt aber in der zweiten Jahreshälfte sehr stark an. Viele der Anfragen kamen aus asiatischen Ländern wie China (16,1%), Südkorea (13,3%), Taiwan (8,4%), Vietnam (6,4%) und Russland (4,6%), wobei sich dies über größere Adressbereiche erstreckt, wie Abbildung 6.3a zeigt.

Die Betrachtung des übertragenen Payload gibt Aufschluss darüber, wie versucht wurde, diesen Exploit auszunutzen. 42,4% der Pakete enthielten lediglich das Byte `0x0a`. Dabei

[1] `https://www.rapid7.com/db/modules/exploit/linux/misc/netcore_udp_53413_backdoor`

[2] `https://www.seebug.org/vuldb/ssvid-90227`



(a) Quelladressen

(b) Zeitliche Entwicklung

Abbildung 6.3.: Darstellungen des Netcore Datenverkehrs

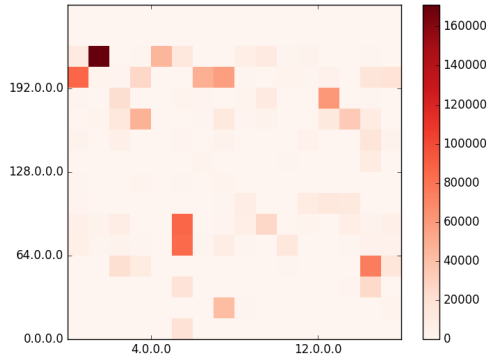
handelt es sich vermutlich um einen einfachen Scan, ob sich hinter der Adresse ein Netcore-Router befindet, welcher auf diese Art der Anfragen antwortet. Der zweit häufigste Payload beinhaltet in ASCII-Darstellung den Wert `AAAAAAAAnetcore`. Ausgehend vom Code, welcher die Schwachstelle im Router ausnutzt[3], handelt es sich dabei um eine Authentifizierung.
Offenbar ermöglichte die Schwachstelle die Ausführung beliebiger Shell-Befehlen auf den betroffenen Geräten. In vielen Paketen ließen sich entsprechende Befehle finden, welche über tftp (`cd /tmp;busybox tftp -g -r m.sh 185.130.5.201`) oder wget (`AA??AAAA cd /tmp || cd /var/; wget http://185.142.236.215/bins.sh`) versuchen, weiteren Code nachzuladen und zur Ausführung zu bringen. Die angeforderten Dateien konnten zum Zeitpunkt der Auswertung nicht mehr an diesen Adressen gefunden werden.
Die ersten acht Bytes einer Nachricht spezifizieren anscheinend die Art der Nachricht. Bei den Authentifizerungs-Paketen sind alle Bytes auf den Wert `0x41` gesetzt. Die Pakete mit Shell-Befehle enthalten zwei Bytes `0x41`, gefolgt von zwei 0-Bytes und vier weiteren `0x41`

6.2.3. Session Initialisation Protocol (SIP)

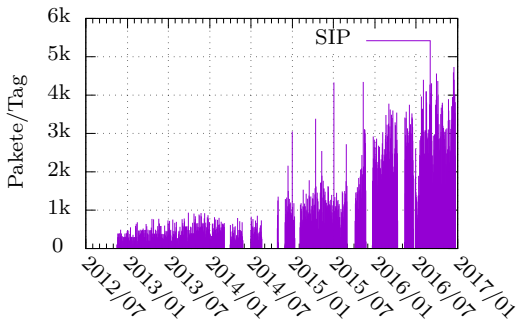
Das Protokoll SIP [RFC3261] wird im Umfeld von Voice-over-IP Telefonaten zur Etablierung von Telefongesprächen eingesetzt. Im Protokoll werden dazu die Parameter der Audio-Übertragung ausgehandelt, etwa die dazu zu verwendenden Ports, Audio-Encodings, aber auch die “Telefonnummer” der Teilnehmer. Die Daten werden als ASCII-Text

[3] `https://github.com/h00die/MSF-Testing-Scripts/blob/master/netis_backdoor.py`

übertragen und sind in ihrem strukturellem Aufbau dem Hypertext-Transfer-Protokoll ähnlich. Wie bei diesem können in SIP User-Agent-Informationen versendet werden.



(a) Quelladressen



(b) Zeitliche Entwicklung

Abbildung 6.4.: Darstellungen des SIP-Datenverkehrs

Bei über 90% des eingehenden SIP-Datenverkehrs ist der Wert des User-Agents auf `friendly-scanner` gesetzt, wodurch eine Einordnung dessen als Scan möglich ist. Dass 94,4% der Anfragen die Methode `OPTIONS` enthalten, unterstützt diese These. Dieser Anfragetyp ist für die Abfrage der unterstützten Funktionen vorgesehen. Weitere 3,8% der Anfragen verwenden die `INVITE` Methode. Diese initiiert eine neue Sitzung. Die Anfragen kommen primär aus den USA (46,7%), Deutschland (20,3%) und Frankreich (12,0%), wobei sich dies über einen größeren Teil des Adressraums verteilt, wie in Abbildung 6.4a zu sehen ist. Häufig vorkommende Adressbereiche ließen sich auf Hosting Provider wie *GoDaddy* und *HostEurope* zurückführen. Seit Beginn der Messung ist das Volumen des SIP-Datenverkehrs von 2012 unter 1000 Paketen pro Tag auf 3000 bis 5000 Pakete pro Tag im Jahr 2016 gestiegen. Ein starker Anstieg ist seit Beginn des Jahres 2015 erkennbar (Abbildung 6.4b).

6.2.4. Port 8000

Bei den auf Port 8000 empfangenen Paketen handelt es sich um fragmentierte IP-Pakete, welche jeweils Daten mit einer Größe von 14.400 Bytes übertragen haben. Die Pakete wurden von einer einzelnen Adresse (106.39.253.238) gesendet, welche ihren Ursprung in China hat. Gesendet wurden die Pakete an zwei Tagen, am 14.10.2015 und am 27.10.2015. Innerhalb weniger Sekunden wurde jeweils mehrere Tausend mal das gleiche Paket gesendet. Die übertragenen Daten konnten jedoch keinem bekannten Pro-

tokoll zugeordnet werden. Die auf Port 8000 empfangenen UDP-Pakete sind für 98,2% der fragmentierten IP-Pakete verantwortlich.

6.2.5. Payload

Nach Filterung der bisher analysierten UDP-Ports werden die übertragenen Daten des übrigen Datenverkehrs genauer betrachtet. Die am häufigsten vorkommenden 8-Byte-Präfixe sind in Tabelle 6.7 zusammengefasst. Es sind vor allem drei Klassen klar erkennbar, BJNP, `d1:ad2:i` und OPTIONS. Bei BJNP handelt es sich um ein Protokoll zur Steuerung von Canon Druckern[4], welches durch die ersten vier Byte eindeutig identifiziert werden kann. Die Kodierung von `d1:ad2:i` entspricht der BitTorrent Erweiterung *DHT Protocol*[5], welche eine auf UDP basierende Alternative zur Nutzung eines zentralen Trackers darstellt. Bei den `OPTIONS` Paketen handelt es sich um den zuvor beschriebenen SIP-Datenverkehr, welcher zu anderen Ports als dem Standardport 5060 gesendet wurde.

Daten (HEX)	Daten (ASCII)	Pakete	Bytes
424a4e5001010000	BJNP????	1.527.586	24.441.376
64313a6164323a69	d1:ad2:i	1.355.606	90.133.668
4f5054494f4e5320	OPTIONS	647.569	266.306.592
0000000000000000	????????	148.559	4.727.408
0600ff0700000000	????????	81.454	1.876.641
0a	?	73.790	73.790
0101000000000000	????????	56.998	1.378.796
0000	??	56.645	113.290
c802006b00000000	???k????	56.208	6.014.256
6000000000003b15	‘?????;?	51.924	2.867.564

Tabelle 6.7.: Präfixe der übrigen über UDP übertragenen Daten

6.3. TCP

Zur Klassifikation des TCP-Datenverkehrs wurden zunächst zusammengehörige Pakete zu Verbindungen zusammengesetzt. Dadurch ist es ebenfalls möglich, die übertragenen Daten zu analysieren, welche über mehrere Pakete aufgeteilt sein können. Auf Grundlage des verwendeten Klassifikationsschemas wurden die Verbindungen in die Kategorien Backscatter, BitTorrent, Scan und Andere unterteilt, wobei die Scans nochmals danach getrennt wurden, ob diese durch einen Fingerprint der Verbindung oder durch die Charakteristiken eines Portscans erkannt wurden. Die Ergebnisse der Klassifizierung sind in Tabelle 6.8 zusammengefasst.

[4] `https://www.wireshark.org/docs/dfref/b/bjnp.html`
[5] `http://www.bittorrent.org/beps/bep_0005.html`

Typ	Verbindungen	Pakete	Bytes	% Verb.	% Pakete	% Bytes
Backscatter	9.503.164	12.642.731	8.979.244	1,8	0,7	0,0
BitTorrent	260.669.337	1.144.223.716	17.735.659.538	48,1	66,2	69,0
Scan (FP)	82.327.285	128.533.727	16.157.523	15,2	7,4	0,1
Scan (Port)	27.006.711	74.423.541	0	5,0	4,3	0,0
Andere	162.146.933	369.393.517	7.945.728.196	29,9	21,4	30,9
Summe	541.653.430	1.729.217.232	25.706.524.501	100,0	100,0	100,0

Tabelle 6.8.: Klassifizierung des TCP-Datenverkehrs

Auffällig ist hierbei zunächst, dass etwa die Hälfte der Verbindungen und über zwei Drittel der Pakete sowie des übertragenen Payloads dem Protokoll BitTorrent zugeordnet werden können. Dies ist für Messungen mittels eines *Blackholes* ungewöhnlich. Hier soll noch die Frage geklärt werden, ob dieser Traffic sich auf den bereits bekannten Implementierungsfehler in der Bibliothek `libtorrent` [libtorrent] zurückführen lässt oder noch weitere Faktoren einen Einfluss auf dieses Phänomen haben.

Der durch Scans verursachte Datenverkehr erzeugte etwa 20% der aufgezeichneten TCP-Verbindungen. Auf diesen Verbindungen wurden jedoch kaum Daten ausgetauscht. Für die erkannten Port- und Stealth-Scans liegt dies in der Art der Klassifizierung begründet. Der Portscan wird unter anderem dadurch klassifiziert, dass über die Verbindung keine Daten gesendet werden, sondern nur geprüft wird, ob der Port offen ist. Beim Stealth-Scan werden ebenfalls keine Daten versendet, da die Verbindung nicht vollständig aufgebaut wird. Die wenigen Daten, die über als Scan erkannte Verbindungen versendet wurden, sind daher vermutlich aufgrund des Scanner-Fingerprints von Masscan oder Zmap klassifiziert worden. Diese These soll bei einer genaueren Betrachtung dieser Kategorie untersucht werden.

1,8% der Verbindungen wurden als Backscatter erkannt, welcher ebenso wie die Scans kaum zur Übertragung von Daten genutzt worden. Für diese Verbindungen soll geprüft werden, ob Ziele von DoS-Angriffen erkannt werden können.

Die übrigen Verbindungen konnten durch das Klassifizierungsverfahren noch nicht klar identifiziert werden. Über diese Verbindungen wurden jedoch auch viele Daten übertragen, sodass eine Betrachtung des Payloads dazu geeignet ist die Daten weiter zu klassifizieren.

6.3.1. BitTorrent

Für die Auswertung des BitTorrent Datenverkehrs wird der über die Verbindungen übertragene Payload genauer betrachtet. Über den Großteil der Verbindungen wurden insgesamt jeweils 68 Byte Nutzdaten übertragen. Dies entspricht der Länge des im *peer wire*

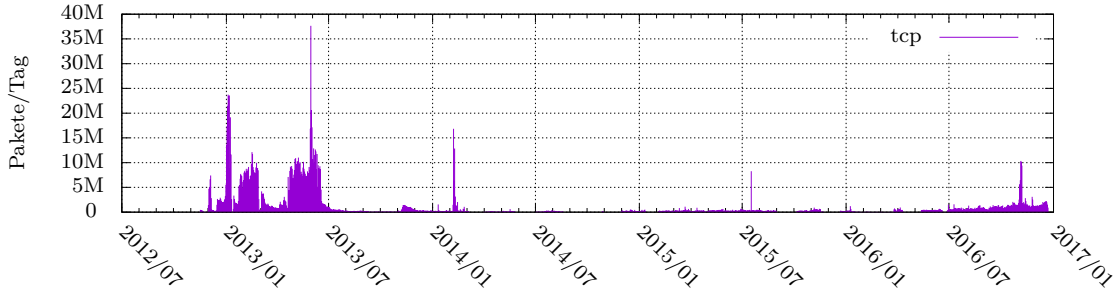
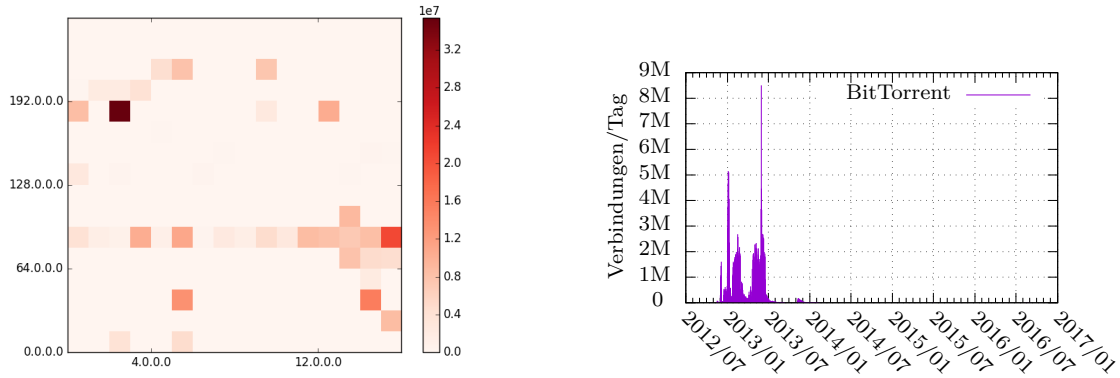


Abbildung 6.5.: Zeitliche Entwicklung der eingegangenen TCP-Pakete

protocol üblichen Verbindungsaufbaus. Diese 68 Byte setzten sich aus vier Komponenten zusammen. Die ersten 20 Byte sind die zur Klassifikation von BitTorrent verwendete statische Kennung. Der übrige *Handshake* beinhaltet acht Byte für Protokollerweiterungen sowie je 20 Byte für den `info_hash` und die `peer_id`. Der `info_hash` wird von Clients über den Meta-Informationen von Torrent-Files gebildet und stellt damit ein Merkmal dar, über welches Rückschlüsse auf die Daten möglich sind, welche der BitTorrent-Client tauschen möchte. Die `peer_id` wird von vielen Implementierungen anhand eines Schemas vergeben, durch welches der verwendete Client identifiziert werden kann. Eine detaillierte Aufschlüsselung der häufigsten Merkmale ist in Tabelle C.2, Tabelle C.3, und Tabelle C.4 aufgeführt.

99,91% des BitTorrent Datenverkehrs beinhaltet einen der 20 häufigsten `info_hashes`. 18 dieser Werte lassen sich durch die Verwendung von Web-Suchmaschinen wie *Google Search* und *Duck Duck Go* einem Torrent zuordnen, zu welchem Meta-Informationen wie Name und enthaltene Dateien verfügbar sind. Das häufigste vorkommende Namensschema ist dabei `wot_8x***_client.patch`. Dieses Namensschema wird vom Computerspiel *World of Tanks* genutzt, dessen Client-Software mittels BitTorrent aktualisiert wird. Dieser Update-Client verwendet die Kennung `-WG0F90-` am Anfang jeder `peer_id`[6], sowie undokumentierte Erweiterungen, welche durch Flags innerhalb der *Reserved Bytes* des Handshakes signalisiert werden. Der Wert dieser acht Bytes wird dort auf `0x0000000000180004` gesetzt. Während die für die Ziffern `1` und `4` verantwortlichen Bits auf die Verwendung des *Libtorrent Extension Protocol (LTEP)* sowie der *Fast Extension* hindeuten, gibt es keine öffentlich bekannte BitTorrent Erweiterung, welche das für die Ziffer `8` verantwortliche Bit verwendet. Vermutlich verwenden die Entwickler von *World of Tanks* eine eigene Protokollerweiterung.

[6] `http://vote.vuze.com/forums/170588-general/suggestions/3500006-add-the-wot-client-string-to-recognized-torrent-cl`



(a) Quelladressen

(b) Zeitliche Entwicklung

Abbildung 6.6.: Darstellungen des BitTorrent Datenverkehrs

Die Quell-IP-Adressen des BitTorrent Datenverkehrs sind zu 70% russischen Bereichen des Adressraums zugeordnet, weitere 20% der Ukraine und Weißrussland. Entsprechend ist auch auf der Heatmap, welche die Verteilung der Adressen darstellt (Abbildung 6.6a) eine Clusterbildung in diesen Adressbereichen erkennbar. Die BitTorrent Anfragen traten primär im Jahr 2013 auf, wie sich aus dem zeitlichen Verlauf in Abbildung 6.6b erkennen lässt. Seit dem Jahr 2014 ist der Anteil von BitTorrent Datenverkehr signifikant zurückgegangen, wobei bis zum Ende der Aufzeichnung immer noch einige wenige Anfragen empfangen wurden. Ursächlich hierfür können noch im Umlauf befindliche Fehlimplementierungen, aber auch Scans nach potentiellen BitTorrent-Peers sein.

Die These, dass der BitTorrent Datenverkehr primär auf den bekannten Implementierungsfehler in der Bibliothek *libtorrent* zurückzuführen ist, lässt sich nach dieser Analyse grundsätzlich bestätigen. Zum einen ist durch die Betrachtung der reservierten Bytes bekannt, dass der Großteil der Anfragen von Clients stammt, die das *Libtorrent Extension Protocol* unterstützen und daher vermutlich auf *libtorrent* basieren, auch wenn eine angepasste Client-Identifikation in der Peer-Id verwendet wird. Weiterhin geht der Anteil an BitTorrent Datenverkehr in der Aufzeichnung einige Monate nach Release der gepatchten *libtorrent*-Version zurück. Das ist eine plausible Zeit, bis zu welcher der Herausgeber von *World of Tanks* diese Version in die eigene Software übernommen hat.

Durch die vor dem Messnetzwerk eingerichteten Filterregeln soll eigentlich verhindert werden, dass das BitTorrent Protokoll im Netz der Universität genutzt werden kann. Die dazu eingerichtete Filterregel (Anhang A), welche die TCP-Ports 6881 – 6889 sperrt, ist dazu offensichtlich jedoch ungeeignet. Das BitTorrent Protokoll sieht diesen Portbereich grundsätzlich vor, kann jedoch auch auf anderen Ports betrieben werden. Ein effektiverer Filtermechanismus für BitTorrent-Datenverkehr kann beispielsweise auf Basis der

übertragenen Daten erfolgen. Die festen 20 Byte zu Beginn des Handshakes, anhand welcher hier auch die Klassifikation des Datenverkehrs vorgenommen wurde, kann hierzu als Filterkriterium genutzt werden.

6.3.2. Scans

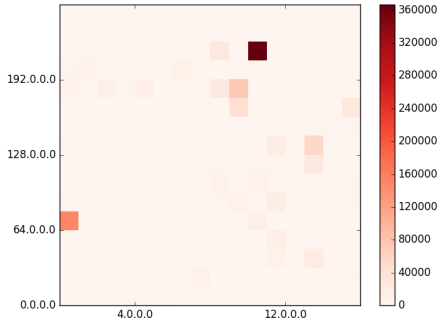
Der durch Scans entstandene Datenverkehr ist abhängig davon, durch welches Merkmal er identifiziert wurde, in vier Bereiche eingeteilt. Dies sind die Fingerprints der Scansoftware *zmap* und *masscan*, sowie die Techniken *Stealth*-Scan und Portscan. Die Anzahl der Verbindungen, die durch das jeweilige Merkmal klassifiziert wurden, unterscheidet sich signifikant. Wie in Tabelle 6.9 zu sehen ist nur bei 2,2% der erkannten Scans der Fingerprint einer der beiden Scanprogramme zu finden. Der mit 73% überwiegende Teil der Scans wurde aufgrund des eingesetzten *Stealth* Verfahrens erkannt, weitere 24% als Portscan. Der Anteil von *Zmap* und *Masscan* kann höher sein, als bei der Auswertung der Daten gemessen wurde. Da beide Programme quelloffen sind, kann der Anwender das einfach zu erkennende Fingerprint-Merkmal entfernen, um zu vermeiden, dass der Scan dadurch einfach identifizierbar und damit filterbar wird. Über die als Portscan erkannten Verbindungen wurden keine Daten übertragen. In 1027 der über 80 Millionen Stealth-Scans wurde ein Payload gefunden, obwohl der Verbindungsaufbau nicht abgeschlossen wurde. Es ist möglich, dass bei diesen wenigen Verbindungen das `ACK` Paket verloren gegangen und nicht erneut übertragen worden ist, sodass es sich dabei nicht um Stealth-Scans handelt. Über 93,7% der Masscan-Verbindungen und 99,9% der Zmap-Verbindungen wurden ebenfalls keine Daten übertragen.

Zmap		Masscan		Stealth		Portscan	
Port	**Verb.**	**Port**	**Verb.**	**Port**	**Verb.**	**Port**	**Verb.**
80	134.974	3389	90.696	80	43.469.900	5900	7.224.214
22	124.051	22	89.033	23	13.181.304	23	6.533.242
443	85.221	80	83.510	21	5.946.205	22	2.225.932
8080	51.832	8080	63.949	22	2.356.189	3389	2.101.157
23	46.535	23	51.005	2323	1.511.276	5901	900.589
21	41.622	443	43.896	3389	825.479	1433	687.097
3389	36.619	21	28.459	7547	570.995	80	561.363
3128	30.576	5900	28.342	1935	561.955	2222	560.433
1604	29.779	3306	17.967	5060	522.990	3306	543.811
502	25.346	110	16.861	5900	522.849	60096	294.069
Gesamt	1.115.479	Gesamt	1.362.386	Gesamt	80.077.372	Gesamt	27.006.711

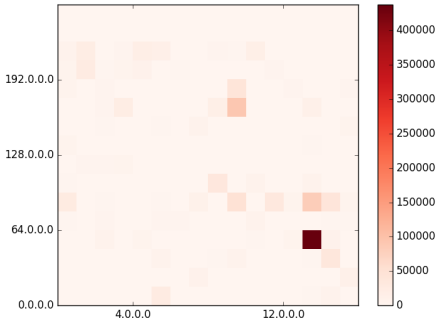
Tabelle 6.9.: 10 häufigste TCP-Zielports der erkannten Scans

Die von den einzelnen Scan Klassen betroffenen Zielports unterscheiden sich teilweise voneinander, wobei Protokolle wie HTTP (80), SSH (22) und Telnet (23) bei allen vier Klassen unter den zehn häufigsten Zielen vorhanden sind. Innerhalb der Scans auf Port 80 wurden nur wenige HTTP-Anfragen gesendet. Die wenigen gestellten Anfragen

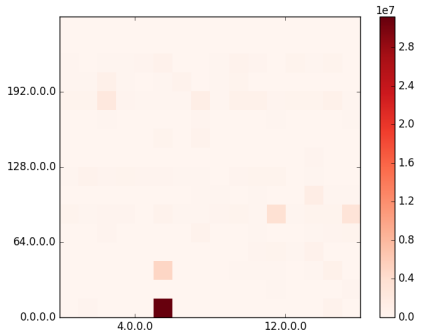
fragen primär das Wurzelverzeichnis ab (`GET / HTTP/1.0`) ab. Sehr wenige Anfragen beziehen sich auf die Verfügbarkeit offener Verwaltungswerkzeuge wie *phpMyAdmin* oder die Wordpress Login-Seite.



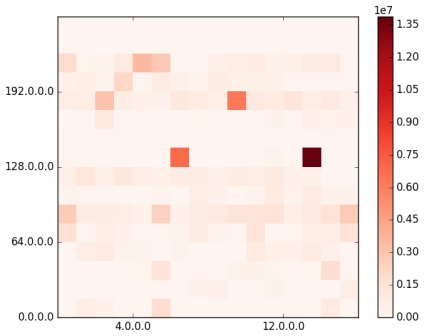
(a) Zmap



(b) Masscan



(c) Stealth-Scan

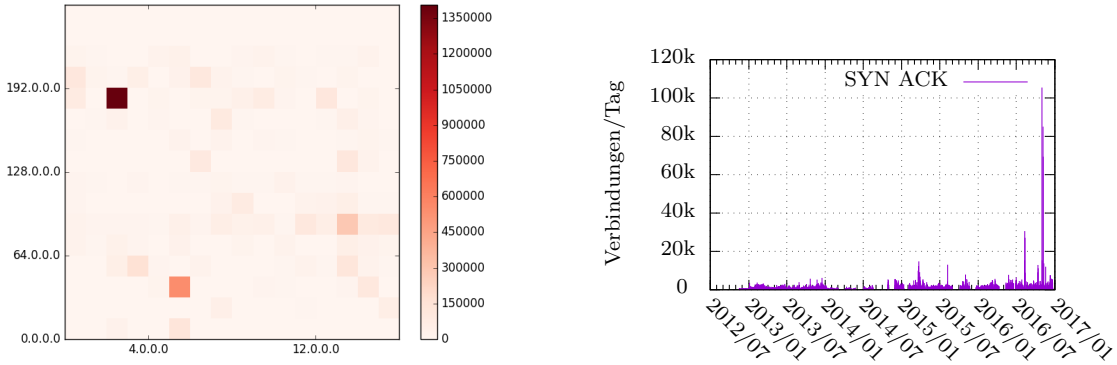


(d) Port-Scan

Abbildung 6.7.: Vergleich der Quelladressen des Scan Datenverkehrs

6.3.3. Backscatter

Backscatter Datenverkehr hat mit 0,7% empfangenen TCP-Pakete den kleinsten Anteil der untersuchten Klassen. Diese bestehen zu 56% aus Verbindungen, welche durch ein `SYN+ACK` initiiert wurden, ohne dass ein `SYN` aufgezeichnet wurde. Die übrigen 44% wurden durch ein `RST` Paket ohne zugehörige Verbindung erkannt. Der in beiden Gruppen am häufigsten angegriffene Port ist 80. Wie in Abbildung 6.8a erkennbar ist, stammt der Backscatter Datenverkehr aus verschiedenen Adressbereichen. Es wurden demnach die Artefakte von DoS Angriffen auf verschiedene Ziele aufgezeichnet. Im zeitlichen Verlauf (Abbildung 6.8b) ist erkennbar, dass insbesondere in der zweiten Jahreshälfte 2016 zwei DoS Angriffe mit gespooften IP-Adressen des Messnetzwerks durchgeführt wurden.



(a) Quelladressen

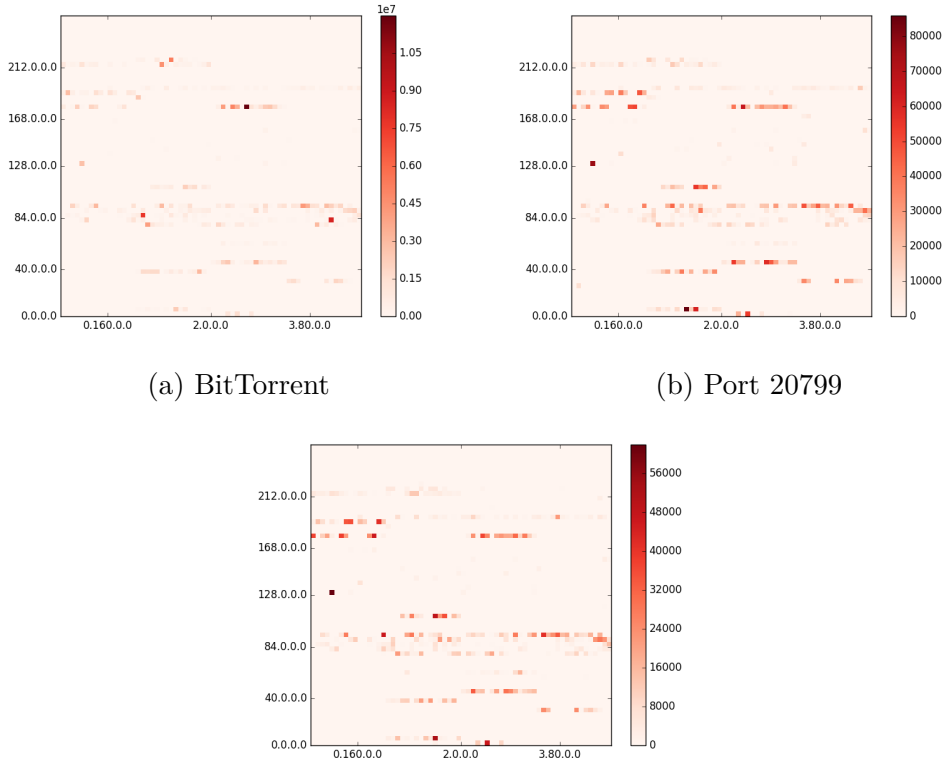
(b) Zeitliche Entwicklung

Abbildung 6.8.: Darstellungen des Backscatter Datenverkehrs

6.3.4. Andere

Ein Teil der bislang nicht klassifizierten Verbindungen ist durch eine technische Besonderheit des Rekonstruktionsverfahrens der TCP-Verbindungen entstanden. Eine TCP-Verbindung wird von der eingesetzten Bibliothek in zwei Fällen als abgeschlossen betrachtet. Der erste Fall ist die Verarbeitung eines TCP-Pakets mit gesetztem `FIN` oder `RST` Flag. Ebenso ist es möglich, dass für die Verbindung über längere Zeit keine Pakete mehr angekommen sind. Über ein *Sliding-Window*-Verfahren werden so offene Verbindungen regelmäßig abgeschlossen. Treffen anschließend weitere Pakete für eine abgeschlossene Verbindung ein, können diese der Verbindung nicht mehr zugeordnet werden. Solche Pakete werden gesondert protokolliert und gezählt. Durch vorherige Filterung lässt sich die Anzahl dieser Pakete bestimmen und aus den Daten extrahieren. Diese Artefakte sind für rund 60% der noch unklassifizierten Verbindungen und 0,1% des übertragenen Payloads verantwortlich. Dieser geringe Anteil an übertragenen Daten kann durch *Retransmissions* entstanden sein.

Über die übrigen 64.112.263 Verbindungen wurden über 7 GB Daten übertragen. Auffällig ist, dass die zwei Zielports dieser Verbindungen, über welche die meisten Daten übertragen wurden (20799 und 48457), keinen öffentlich dokumentierten Anwendungsfall haben. Der übertragene Payload enthält ebenfalls keine erkennbare Struktur. Bei Betrachtung der IP-Adressen, von welchen diese Verbindungen ausgingen, fällt jedoch auf, dass eine Korrelation zwischen den Quelladressen dieser beiden Ports und denen des zuvor klassifizierten BitTorrent Traffics (Abbildung 6.9) besteht. Die Zieladressen konzentrieren sich bei beiden Ports stark auf eine einzelne Adresse innerhalb des Messnetzwerks.



(a) BitTorrent

(b) Port 20799

(c) Port 48457

Abbildung 6.9.: Vergleich der Quelladressen von BitTorrent Datenverkehr und häufigen unklassifizierten Zielports

Bei einer manuellen Betrachtung eines Ausschnitts der betroffenen PCAP-Dateien lässt sich feststellen, dass auf den Ports 20799 und 48457 kurz vor den unklassifizierten Daten eine Verbindung mit einem BitTorrent Handshake der selben Quell-Adresse stattgefunden hat. Nach Ende dieser Verbindung wird eine neue mit gleichem Zielport etabliert. Dies bestätigt die Vermutung, dass dieser Datenverkehr in Zusammenhang mit bekannten BitTorrent Verbindungen entstanden ist. Das BitTorrent-Protokoll gibt jedoch einen formalen Aufbau von Nachrichten vor, welchem diese Daten nicht entsprechen.

Es existiert mit der *Message Stream Encryption (MSE)*[7] eine Möglichkeit um BitTorrent Datenverkehr zu verschleiern. Da BitTorrent aufgrund des Handshakes einfach identifiziert und gefiltert oder gedrosselt werden kann, wurde diese Erweiterung geschaffen, um Filtermaßnahmen zu umgehen. MSE benötigten zu Beginn einen kryptografischen Schlüsselaustausch über das Diffie-Hellman-Verfahren. Die initiale Nachricht beinhaltet eine 768 Bit (96 Byte) lange Zahl zur Schlüsselgenerierung sowie einen zufälligen Wert mit einer zufälligen Länge von 0 bis 512 Byte. Dadurch unterscheidet sich die Länge der Nachrichten oft, liegt aber immer im Bereich zwischen 96 und 608 Byte. Die auf den zwei Ports empfangenen Daten schwanken in ihrer Länge in diesem Bereich, wie in Abbildung 6.10 zu sehen ist. Es handelt sich dabei also höchstwahrscheinlich um verschleierten BitTorrent Datenverkehr.

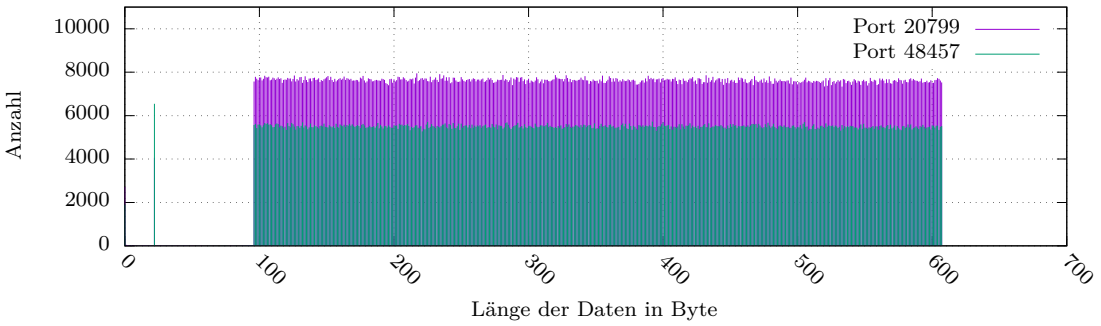


Abbildung 6.10.: Länge der Nachrichten auf Port 20799 und 48457

Unter den häufig kontaktierten Zielports (Tabelle 6.10) finden sich weiterhin bekannte Anwendungsprotokolle wie Telnet (23), SSH (22), das *Remote Desktop Protocol* (3389), HTTP (80) und das *Remote Framebuffer Protocol* (5900). Vier dieser Protokolle sind grundsätzlich dazu ausgelegt die direkte entfernte Steuerung eines Computers zu ermöglichen. Ob die eingegangenen Verbindungen auf Versuchen beruhen, diese Steuerung durchzuführen, wird eine Betrachtung der übertragenen Daten ergeben.

[7] `https://wiki.vuze.com/w/Message_Stream_Encryption`

10 häufiste Ports nach Verbindungsanzahl					10 häufigste Ports nach Payload				
Port	**Verb.**	**%**	**Bytes**	**%**	**Port**	**Verb.**	**%**	**Bytes**	**%**
23	13.123.917	20,5	8.607.142	0,1	20799	3.914.784	6,1	1.370.895.309	17,3
22	6.891.830	10,8	219.278.602	2,8	48457	2.825.800	4,4	989.204.188	12,5
3389	5.542.478	8,6	255.247.354	3,2	7547	395.510	0,6	293.989.289	3,7
20799	3.914.784	6,1	1.370.895.309	17,3	3389	5.542.478	8,6	255.247.354	3,2
48457	2.825.800	4,4	989.204.188	12,5	9200	403.444	0,6	239.586.279	3,0
1433	2.053.824	3,2	163.596.993	2,1	5402	801.320	1,3	232.236.888	2,9
80	1.673.699	2,6	210.847.389	2,7	21320	767.968	1,2	227.095.102	2,9
5900	1.175.201	1,8	5.102.304	0,1	59418	713.981	1,1	222.860.826	2,8
2323	974.205	1,5	2.777.597	0,0	22	6.891.830	10,8	219.278.602	2,8
61978	815.286	1,3	135.552.300	1,7	80	1.673.699	2,6	210.847.389	2,7
Summe	64.112.263	60,8	7.936.360.657	42,4		64.112.263	37,3	7.936.360.657	53,7

Tabelle 6.10.: 10 häufigste TCP-Zielports aufgeschlüsselt nach Anzahl und Payload

6.3.4.1. Fernzugriffsprotokolle

Die häufig vorkommenden Fernzugriffsprotokolle erlauben grundsätzlich erst nach erfolgreicher Authentifizierung eines Benutzers die Steuerung des Computers. Beim *Remote Framebuffer Protocol* auf Port 5900 lassen sich jedoch keine Authentifizierungsversuche feststellen. Dies liegt daran, dass das Protokoll zunächst eine vom Server gesendete Versionskennung vorsieht. Da diese vom verwendeten *active Responder* nicht versendet wird, erfolgt keine weitere Nachricht des Clients.
Ähnlich verhält es sich beim SSH-Protokoll (Port 22), welches vor der Authentifikation den gegenseitigen Austausch einer "Hello" Nachricht vorsieht. Anstelle von Authentifikationsversuchen lassen sich im Payload der SSH-Verbindungen daher Kennungen der Clients finden, welche Aufschluss über die verwendete Software bieten. Auf knapp 30% der Verbindungen wurde die Kennung des Clients *PuTTy*[8] gesendet, welcher hauptsächlich im Windows-Umfeld zum Einsatz kommt. Die für Linux Distributionen übliche libssh2 ist in den Versionen 1.4.1 (16%), 1.6.0 (7,6%) und 1.7.0 (5,5%) seltener vertreten.

Beim *Remote Desktop Protocol* auf Port 3389 sind die Authentifizierungsversuche als Administrator (`Cookie: mstshash=Admin`) klar erkennbar. Beim Telnet Protokoll (Port 23) werden auf 98% der Verbindungen keine Daten übertragen. In den übrigen Verbindungen sind ebenfalls Authentifizierungsversuche für privilegierte Nutzer wie root und admin mit Standardpasswörtern erkennbar. Vor allem beim Telnet Protokoll lässt sich ein starker Anstieg der Verbindungen im Jahr 2016 feststellen (Abbildung 6.11)

[8] `http://www.putty.org/`

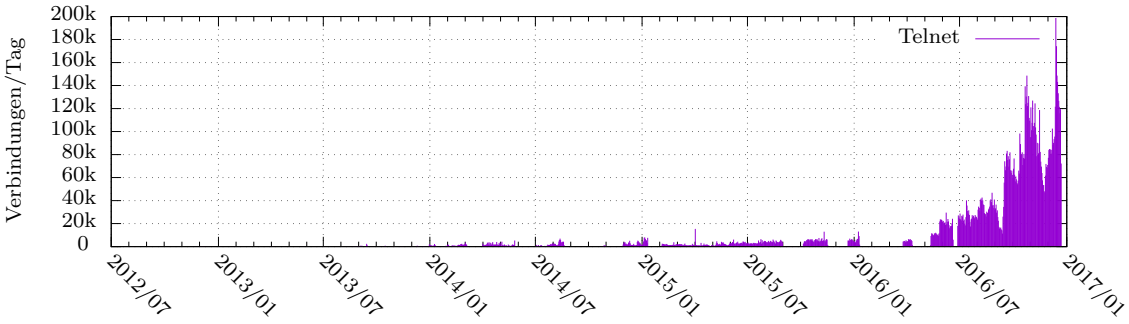


Abbildung 6.11.: Zeitlicher Verlauf der Telnet Verbindungen

6.3.4.2. HTTP und HTTPS

Die über HTTP eingegangenen Anfragen entsprechen zum Großteil den Charakteristiken eines einfachen Scans. 67,2% der Anfragen verwenden die Zugriffsmethode `GET`, weitere 12,7% `HEAD` (Tabelle C.5). Diese dienen jeweils nur zur Abfrage von Informationen, ohne Modifikationen an den Daten vorzunehmen. Die häufigste angefragte Ressource ist dabei mit 27,9% das Wurzeldokument. Diese Anfragen wurden demnach nicht mit einem speziellen Ziel gestellt. Anders verhält es sich bei den übrigen Dokumenten, welche häufig angefragt wurden. In den TOP 10 (Tabelle C.6) befinden sich vor allem die Standardadressen von Managementwerkzeugen wie der Datenbankverwaltung *phpMyAdmin*. Dies deutet darauf hin, dass gezielt nach frei zugänglichen Instanzen dieser Anwendung gescannt wurde. Durch eine Modifikation der Messung, die entsprechende HTTP-Anfragen positiv beantwortet, könnte zukünftig festgestellt werden, ob diese Scanner im Anschluss versuchen, bekannte Exploits auf die gefundene Instanz von *phpMyAdmin* anzuwenden.

Auffällig sind weiterhin die 3,1% Anfragen mit der Zugriffsmethode `CONNECT`, sowie die 2,06% welche die `POST` Methode verwenden. Die erste ist dafür vorgesehen über HTTP einen Tunnel zu anderen auf TCP basierenden Protokollen aufzubauen. In den aufgezeichneten Daten befinden sich Anfragen wie `CONNECT mx2.mail2000.com.tw:25`, bei welchen über diesen Mechanismus versucht wird eine Verbindung zu einem Mailserver herzustellen. Bei diesen Anfragen handelt es sich vermutlich um die Suche nach sogenannten *Open Relays*, also Proxy Servern, welche ohne vorherige Authentifizierung genutzt werden können. Bei den empfangenen `POST` Requests wurde teilweise explizit versucht bekannte Sicherheitslücken, etwa in der Laufzeitumgebung der Programmiersprache PHP, auszunutzen um Zugriff zum System zu erlangen. Anfragen dieser Art sind etwa `POST /cgi-bin/php...-d+auto_prepend_file=php://input+...`. Dabei wird versucht di-

verse Sicherheitsparameter auszuschalten um dann den in der Anfrage mitgeschickten PHP-Code zur Ausführung zu bringen.

Für HTTPS können diese Auswertungen des Datenverkehrs nicht vorgenommen werden. Da der *active responder* lediglich auf TCP-Anfragen antwortet, gibt der auf Port 443 empfangene Datenverkehr weniger Aufschluss über die Intention der Anfrage, als beim ungeschützten HTTP. Untersuchen lassen sich jedoch die Parameter des kryptografischen Verbindungsaufbaus. Tabelle C.7 fasst die verwendeten Nachrichtentypen der auf Port 433 eingegangen Daten zusammen. Der *Handshake*, welcher zur Initiierung der TLS-Verbindung dient ist dabei mit einem Anteil von 52,2% der häufigste vorkommende Typ. Weitere 45,3% enthalten zwar Daten, welche jedoch nicht dem im TLS-Protokoll vorgegebenen Schema entsprechen. Auf 2,4% der Verbindungen wurden keine Daten übertragen. Auch wenn auf manchen der nicht TLS-konformen Verbindungen versucht wurde, Daten im Klartext zu übertragen, kann eine vollständige Analyse der Daten nur sichergestellt werden, wenn zusätzlich zum aktiven Antworten von TCP-Verbindungsanfragen auch TLS *Handshakes* beantwortet werden.

Die tatsächlich als TLS identifizierten Verbindungen basieren zu 89% auf der Protokollversion TLS 1.0 (Tabelle C.8). Die derzeit aktuelle Version TLS 1.2 wird nur bei 2% der Verbindungsanfragen eingesetzt. Bei den empfangenen Handshakes handelt es sich zu 99,98% um Pakete des Typs *ClientHello*. *ClientHello* Pakete enthalten unter anderem eine Liste der vom Client unterstützen *CipherSuites*, welche eine Kombination von Algorithmen für kryptografische Operationen beschreiben. Die zwanzig häufigsten *CipherSuites* (Tabelle C.11) enthalten einige, welche nach derzeitigem Kenntnisstand nicht mehr als sicher angesehen werden. Der symmetrische Verschlüsselungsalgorithmus RC4 soll etwa seit Februar 2015 nicht mehr verwendet werden[9]. Da der Messzeitraum jedoch den Zeitraum ab 2012 umfasst, ist es plausibel, dass die gemessenen Werte nicht dem aktuellen Standard entsprechen.
Die Verwendung von Kompression in TLS gilt jedoch seit 2012 als Sicherheitsrisiko. Der als "CRIME"[10] bekannte Angriff ermöglicht etwa das Auslesen sensibler Informationen wie Session-Cookies in durch TLS geschützten Webseiten. Als effektive Schutzmaßnahme gilt die Deaktivierung von TLS-Kompression. In den eingegangenen *ClientHello* Paketen wurde in 90% keine Kompression unterstützt. Bei 9% wurde jedoch der für den Angriff ausgenutzte *DEFLATE* Algorithmus zur Kompression vorgeschlagen (Tabelle C.10).

Die Intention der TLS-Anfragen lässt sich aufgrund der gesammelten Daten nicht zweifelsfrei klären. Es empfiehlt sich daher, die Messung um eine zusätzliche Antwortkom-

[9] `https://tools.ietf.org/html/rfc7465`
[10] `https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2012/september/details-on-the-crime-attack/`

ponente für eingehende TLS-Anfragen zu erweitern. Eine Detektion von TLS kann unabhängig vom Port über das Präfix der ersten übertragenen Bytes erfolgen. Der TLS-Record beginnt immer mit einem Byte, welches den *RecordType* festlegt, gefolgt von zwei Bytes, welche die verwendete Version benennen. Für beide Werte sind wenige fest definierte Ausprägungen im Protokoll vorgesehen.

7. Fazit

Die Analyse der vorliegenden Daten hat gezeigt, dass der Einsatz eines *active responders* starken Einfluss auf die Art und das Volumen der empfangenen Daten hat. Obwohl die Messung für ICMP-Datenverkehr ein vollständig passives Verhalten zeigt, konnten rund 50% des ICMP-Datenverkehrs auf den Einsatz des *active responders* für TCP zurückgeführt werden. Dies muss bei Verwendung dieser Messmethode für die Analyse der aufgezeichneten Daten beachtet werden. Ebenso sind große Teile des TCP-Datenverkehrs, welche das Protokoll BitTorrent beinhalten, durch die Kombination eines Implementierungsfehlers in `libtorrent` und den Einsatz des *active responders* entstanden.

Die Auswertung hat gezeigt, dass die bisherige Filterung von BitTorrent durch Sperrung einzelner Ports unzureichend ist. Eine effektive Filterung kann bei unverschlüsseltem BitTorrent Datenverkehr aufgrund des BitTorrent Handshakes durchgeführt werden. Durch die Erweiterung MSE verschleierter BitTorrent Datenverkehr kann im Nachhinein durch eine Häufigkeitsanalyse auf der Länge der übertragenen Daten erkannt werden. Daraus lässt sich jedoch keine Filterregel ableiten. Die verschleierten Verbindungsanfragen über MSE folgten in vielen Fällen einem zuvor im Klartext gesendeten BitTorrent Handshake. Ob sich aus diesem Zusammenhang ein Filterkriterium für verschleierten BitTorrent Datenverkehr herleiten lässt, kann in einer zukünftigen Untersuchung betrachtet werden.

Die zweit häufigste Klasse von Datenverkehr sind Scanning Aktivitäten. Der durch Scanning verursachte Datenverkehr hat insbesondere für die Protokolle SIP, RDP, Telnet und SSH in den vergangenen Jahren stark zugenommen. Zumeist beschränken sich diese Anfragen bislang auf die Prüfung von Verfügbarkeit der Protokolle oder bestimmter Daten. Jedoch wurden bei der Analyse der übertragenen Daten auch Authentifizierungsversuche und Exploits gefunden.
Im TCP-Datenverkehr ist kaum Backscatter enthalten. Für das DNS-Protokoll ist dieser jedoch deutlich stärker vertreten als Scans nach Open Resolvern.

7.1. Bewertung

Durch das Klassifikationsschema für ICMP wurden der gesamte Datenverkehr dieses Protokolls klassifiziert. Mittels des iterativen Verfahrens konnten 64,2% der eingegangenen UDP-Pakete, bzw. 86,3% der per UDP empfangenen Daten zugeordnet werden (Tabelle C.12). Dazu wurden sieben Iterationen angewendet (Tabelle C.13).
Bereinigt man die erkannten 541 Millionen TCP-Verbindungen (Tabelle 6.8) um die in Unterabschnitt 6.3.4 beschriebenen Artefakte, werden 85,5% dieser Verbindungen automatisch als BitTorrent, Scanner oder Backscatter klassifiziert. Für die übrigen Verbindungen ließen sich durch weitere Iterationen zusätzliche Anzeichen für Scanner Aktivitäten finden. Insgesamt wurden dazu acht Iterationen des Analyseverfahrens auf dem TCP-Datenverkehr angewendet.

Die Anwendung des Iterationsverfahrens auf den Daten der semi-aktiven Messung zeigt, dass dieses bereits nach wenigen Iterationen eine hohe Klassifikationsrate erreicht. Durch weitere Iterationen wird diese jedoch nur noch geringfügig erhöht. Das Verfahren eignet sich somit gut für die Analyse des Großteils der Daten. Um eine effektive Auswertung der übrigen unklassifizierten Daten vorzunehmen sind jedoch weitere Anpassungen sinnvoll. Möglich wäre die Erweiterung des Iterationsverfahrens um zusätzliche Merkmale, welche auf markante Punkte geprüft werden oder die Kombination mit anderen Analyseverfahren.

Die konzipierte Software ermöglicht eine effiziente Durchführung der vorgenommenen Auswertungen. Durch Einsatz von Lesepuffern für das Auslesen der PCAP-Dateien konnte eine Verbesserung der Lesegeschwindigkeit von bis zu 65% gegenüber der Standardbibliothek `libpcap` erreicht werden. Ebenso konnte durch eine vorherige Transformation der Aufzeichnungsdaten die TCP-Rekonstruktion parallelisiert werden. Durch den modularen Aufbau mittels einer Pipeline-Architektur ist ebenfalls eine Erweiterung der Software für andere Analyseansätze möglich.

7.2. Ausblick

Die umgesetzte Analysesoftware wird derzeit durch ein manuelles Editieren der Konfigurationsdatei im XML-Format konfiguriert. Für den Aufbau komplexer Pipelines ist dieses Format jedoch umständlich. Denkbar ist an dieser Stelle eine Erweiterung der Software um eine grafische Konfigurationsschnittstelle, welche die Pipeline in Form eines geeigneten Diagramms darstellt. Dies würde eine einfachere Konfiguration der Software ermöglichen.

Das eingesetzte Analyseverfahren ist wie zuvor beschrieben grundsätzlich für die Auswertung von semi-aktiven Messungen geeignet, erfordert jedoch bisher eine manuelle Betrachtung der Detailergebnisse einzelner Phänomene. Eine mögliche Erweiterung dieses Ansatzes könnte bisherige Klassifikationsergebnisse als Trainingsdaten für ein neuronales Netzwerk verwenden, um anschließend noch unklassifizierte Daten auszuwerten.

Für die Messung des Datensatzes wurde eine Antwortkomponente für das Transportprotokoll TCP verwendet. Die Auswertung der Daten hat gezeigt, dass jedoch weitere Antwort-Komponenten sinnvoll sind, um weitere Informationen über die Intention des Senders zu erfahren. Dies betrifft insbesondere verschlüsselten Datenverkehr. Die Messung sollte um Antwort-Komponenten für das TLS-Protokoll erweitert werden. TLS-Anfragen lassen sich anhand der ersten gesendeten Bytes portunabhängig erkennen. Da TLS für verschiedene Anwendungsprotokolle eingesetzt werden kann, ist die portunabhängige Erkennung einer statischen Zuweisung, etwa für den HTTPS-Port 443, vorzuziehen.

Anhang A.

Gefilterte IP-Adressen und Ports

Quell-IP	Protokoll	Quell-Port	Ziel-Port
81.169.138.14			
61.97.152.3			
143.225.92.26			
85.17.52.82			
	udp	0	
	udp		0
	udp		22
	udp		23
	udp		110
	udp		137
	tcp		137
	udp		138
	tcp		138
	udp		139
	tcp		139
	tcp		445
	udp		445
	tcp		1434
	udp		1434
	udp	1900	
	tcp	6000	
	tcp		4232
	udp		4236
	tcp		4236
	udp		4244
	tcp		4246
	udp		5353
	udp		123
	tcp		1214
	udp		1214
	udp	1214	
	tcp		6868
	tcp		6969
	tcp		9898
	tcp		4881

Quell-IP	Protokoll	Quell-Port	Ziel-Port
	tcp		7881
	tcp		9881
	tcp	!53	6881 – 6889
	tcp	6881 – 6889	!161
	udp		6881 – 6889
	udp	6881 – 6889	!53
	tcp		5634
	udp		5634
	tcp		6346
	udp		6346
	tcp		6347
	udp		6347
	tcp		6348
	udp		6348
	tcp		6349
	udp		6349
	tcp		6355
	udp		6355
	tcp		3410
	tcp		5554
	tcp		6129
	tcp		2745
	tcp		4751
	tcp		6777
	tcp		8866
	tcp		2766
	tcp		3127
	tcp		2967
	tcp	80	
	tcp		1
	udp		1
	tcp		2
	udp		2
	tcp		3
	udp		3
	tcp		4
	udp		4
	tcp		5
	udp		5
	tcp		6
	udp		6
	tcp		8
	udp		8
	tcp		9
	udp		9
	tcp		10
	udp		10
	tcp		11
	udp		11
	tcp		12

Quell-IP	Protokoll	Quell-Port	Ziel-Port
	udp		12
	tcp		13
	udp		13
	tcp		14
	udp		14
	tcp		15
	udp		15
	tcp		16
	udp		16
	tcp		19
	udp		19
	tcp		37
	udp		37
	tcp		42
	udp		42
	tcp		43
	tcp		57
	udp		57
	udp		67
	udp		68
	tcp		79
	udp		81
	tcp		93
	tcp		111
	udp		111
	udp		135
	tcp		135
	tcp		161
	udp		161
	tcp		162
	udp		162
	udp		177
	tcp		199
	udp		199
	tcp		280
	udp		280
	udp		389
	tcp		391
	udp		391
	tcp		411
	tcp		412
	udp		427
	udp		509
	tcp		512
	udp		513
	tcp		514
	udp		514
	tcp		515
	udp		517
	udp		518

Quell-IP	Protokoll	Quell-Port	Ziel-Port
	tcp		524
	tcp		593
	tcp		631
	udp		666
	tcp		705
	tcp		1023
	tcp		1025
	udp		1025
	udp		1026
	udp		1027
	udp		1028
	udp		1029
	tcp		1034
	udp		1040
	tcp		1040
	udp		1046
	tcp		1067
	udp		1067
	tcp		1080
	tcp		1243
	tcp		1285
	tcp		1299
	tcp		1331
	tcp		1337
	udp		1900
	tcp		1905
	tcp		1993
	udp		1993
	tcp		2002
	udp		2002
	tcp		2041
	udp		2049
	tcp		2049
	tcp		2121
	tcp		2234
	udp		2273
	tcp		2352
	udp		2352
	tcp		2800
	udp		2798
	tcp		3067
	tcp		3135
	tcp		3136
	tcp		3137
	udp		3354
	udp		3372
	tcp		3372
	tcp		3374
	udp		3434
	tcp		4045

Quell-IP	Protokoll	Quell-Port	Ziel-Port
	udp		4045
	udp		4093
	tcp		4093
	udp		4444
	tcp		4899
	udp		4999
	udp		5003
	tcp		5101
	tcp		5501
	tcp		5506
	tcp		5534
	tcp		5555
	udp		5555
	tcp		5556
	tcp		5557
	tcp		5661
	tcp		5678
	tcp		5851
	udp		5855
	udp		6112
	tcp		6257
	udp		6257
	udp		6343
	tcp		6588
	tcp		6640
	tcp		6666
	udp		6666
	tcp		6697
	tcp		6677
	tcp		6688
	tcp		6699
	tcp		6700
	tcp		6711
	tcp		6712
	tcp		6713
	tcp		6776
	tcp		7080
	tcp		7112
	tcp		7719
	udp		7719
	tcp		7757
	tcp		7784
	tcp		8001
	tcp		8079
	tcp		8181
	udp		8888
	udp		8875
	tcp		8875
	tcp		8876
	tcp		8888

Quell-IP	Protokoll	Quell-Port	Ziel-Port
	tcp		8899
	tcp		9100
	tcp		9494
	tcp		9495
	tcp		9401
	tcp		9402
	udp		9822
	tcp		9822
	tcp		9876
	tcp		9925
	udp		11112
	tcp		1368
	udp		1368
	tcp		4104
	udp		4104
	tcp		10168
	tcp		12345
	tcp		27374
	udp		27960
	udp		28960
	udp	28960	
	tcp		36794
	udp		36794
	tcp		7729 – 7734
	udp		7729 – 7734
	tcp		27010 – 27020
	udp		27010 – 27020

Anhang B.

Aufbau der Analyse-Konfiguration

Die Implementierung der Software ist als Git Repository unter `https://github.com/jzaeske/pcap-analyser/tree/thesis` verfügbar. Darin enthalten ist eine formale Definition des Konfigurationsformats als XML-Schema unter `https://github.com/jzaeske/pcap-analyser/blob/thesis/analyser/schema/analysis.xsd`

Zur Installation wird go in der Version 1.8 auf dem Zielsystem benötigt[1]. Zur Installtion genügt dann die Ausführung des Befehls `go install github.com/jzaeske/pcap-analyser`. Dadurch werden die benötigten Pakete heruntergeladen und kompiliert. Die ausführbare Datei befindet sich unter `$GOPATH/bin`. Zur Ausführung gebracht wird diese durch `$GOPATH/bin/pcap-analyser -config=config.xml`, wobei config.xml eine Konfigurationsdatei im hier beschriebenen Format ist.

B.1. Verfügbare Komponenten

Alle Komponenten benötigen mindestens das Attribut `id (string)`, über welches die Verknüpfung der Pipeline definiert wird. Weitere Parameter sind im Folgenden aufgelistet.

Parser extrahiert die Protokollschichten aus den ausgelesenen Daten

Composer führt IP Defragmentierung und die Zusammensetzung von TCP-Verbindungen durch

- **keepPayload (bool)** rekonstruierte Daten speichern. Ansonsten wird nur die Größe gespeichert.
- **onlyDefrag (bool)** nur IP Defragmentierung durchführen, keine TCP-Rekonstruktion
- **inEthDst (string)** Ethernet MAC Adresse des Empfängers. Wird zur Korrektur der Richtung verwendet, Pakete in falscher Reihenfolge vorliegen

StreamCounter zählt TCP-Verbindungen, Paketanzahl und übertragene Daten.

- **includeScores (bool)** zählt zusätzlich auf der Verbindung berechnete Scoring-Werte (s. Scorer) mit

PacketCounter zählt Pakete, sowie Größe der Kopfaten und übertragenen Nutzdaten

- **layer (int)** Schicht, auf deren Basis die Trennung von Kopf und Nutzdaten erfolgen soll. Werte sind unter `https://github.com/google/gopacket/blob/master/layers/layertypes.go` festgelegt.

StreamScorer führ das Scoring von TCP-Verbindungen durch. Enthält eine beliebige Anzahl an Scores, wie in Abschnitt B.3 beschrieben

[1] `https://golang.org/doc/install`

Filter filtert Pakete anhand von BPF oder Zeit

bpf (string) BPF-Filterregel

minTime (string) Untergrenze eines Zeitraums im Format YYYY/mm/dd HH:MM:ss

maxTime (string) Obergrenze eines Zeitrausm im Format YYYY/mm/dd HH:MM:ss

drop (bool) Pakete verwerfen, die der Filterregel nicht entsprechen, anstatt über den zweiten Ausgang weiterzusenden

StreamFilter filter Verbindungen anhand vorher berechneter Scores. Enthält eine beliebige Anzahl von *ScoreComparator*

policy (enum) "or" oder "and" legt fest, ob mindestens einer oder alle *ScoreComparator* die Verbindung zulassen.

ScoreComparator vergleicht einen Score gegen einen Wertebereich

score (string) Name des Scores, der verglichen werden soll

min (int) Minimaler Wert, den der Score annehmen muss

max (int) Maximaler Wert, den der Score annehmen darf. (Der Wert '0' muss durch '-1' kodiert werden)

PacketOutput schreibt Pakete in eine PCAP-Datei. Kann durch Klassifikatoren in mehrere Dateien aufgeteilt werden.

fileName (string) Dateiname der PCAP-Datei. Der Platzhalter '$' wird bei Verwendung eines Klassifikators ersetzt.

CsvStreamOutput schreibt Informationen von TCP-Verbindungen in eine CSV-Datei.

outputFile (string) Name der Datei. Endet diese auf .gz, wird die Ausgabe automatisch komprimiert. Der Platzhalter '$' wird durch eine eindeutige Zahl ersetzt, damit mehrere Worker parallel schreiben können. Der Platzhalter '#' wird bei Einsatz eines Klassifikators durch dessen Wert ersetzt.

fields (string) Komma-getrennte Liste der Attribute, welche in die CSV-Datei geschrieben werden. Möglich sind ip und port (jeweils mit Preifx s_ für Server und c_ für Client), t_start für den Startzeitpunkt, t_dur für die Dauer in Millisekunden, payload für die übertragenen Daten (hex) sowie alle berechneten Scores.

B.2. Verfügbare Klassifikatoren

Klassifikatoren werden immer als Kind-Elemente innerhalb der Pipeline-Elemente definiert, in welchen sie genutzt werden.

PortClassifier gibt den Zielport der Verbindung oder Pakets zurück

reverse (bool) alternativ den Quellport

Ip4Classifier gibt die IPv4 Adresse des Ziels zurück

reverse (bool) alternativ die Ipv4 Adresse des Absenders

cidr (int) berechnet das Netz anhand des CIDR und gibt dieses zurück

Ip4PortClassifier Kombination aus PortClassifier und Ip4Classifier

reverseIp (bool) wie reverse beim Ip4Classifier

reversePort (bool) wie reverse beim PortClassfier

cidr (int) wie cidr beim Ip4Classifier

DayClassifier gibt den Zeitpunkt zurück zu welchem das Paket aufgezeichne wurde.

format (string) Formatierungszeichenkette des Referenzzeitpunkts gemäß `https://golang.org/pkg/time/#Time.Format`

PayloadClassifier Gibt hexadezimal kodierten Payload zurück

bytes (int) Anzahl der verwendeten Bytes

offset (int) Startposition ausgehend von 0. Standard ist der Wert 0

StaticClassifier Gibt einen festen Wert zurück. Wird beispielsweise genutzt, wenn die Klassifikation nur in eine Dimension stattfinden soll. Legt dann den Dateinamen fest.

filename (string) der Dateiname (ohne Endung)

IcmpClassifier gibt den ICMP-Type zurück inklusive Kennzeichnung, ob ein *active Responder* Paket im Payload vorhanden ist

TransportClassifier gibt den Wert des *NextHeader* Felder im IP Paket zurück

B.3. Verfügbare Scores

Alle Scores enthalten mindestens zwei Attribute. `identifier (string)` legt den eindeutigen Namen dieses Scores fest, unter welchem der berechnete Wert gespeichert wird. Durch score (string) wird der ein Name angegeben, in welchen mehrere Scores addiert werden.

ZMapFPScore prüft, ob alle eingehenden Pakete der Verbindung dem Fingerprint der Software Zmap entsprechen.

MasscanFPScore prüft, ob alle eingehenden Pakete der Verbindung dem Fingerprint der Software Masscan entsprechen

HandshakeScore prüft, ob die Pakete des Verbindungsaufbaus einem festgelegten Muster entsprechen. Das Attribut `pattern (string)` gibt das Muster als Komma-separierte Liste an. Der erste Wert bezieht sich auf das initiale `SYN` des Verbindungsaufbaus, der zweite auf das folgende `SYN+ACK`, das dritte auf das abschließende `ACK`. Optional kann ein viertes Feld geprüft werden. Dieses wird gesetzt, wenn die ein eigehendes `RST` ohne vorherigen Verbindungsaufbau empfangen wurde. Jeder der vier Werte kann den Wert -1, 0 oder 1 annehmen. 1 bedeutet, dass das Paket vorhanden, sein muss, -1, dass es nicht vorhanden sein darf. Der Wert 0 legt fest, dass das Paket für diese Prüfung nicht relevant ist.

PayloadScore prüft, ob der Payload der Verbindung mit einem festgelegten Wert beginnt. Dieser wird im Attribut `hexVal (string)` definiert. Ist dieser Wert leer, bedeutet dies, dass die Verbindung keine Daten enthalten darf.

DurationScore prüft, ob die Dauer der Verbindung einem festgelegten Zeitrahmen entspricht. Dieser wird durch die Attribute `minMillis (int)` und `maxMillis (int)` in Millisekunden angegeben.

Anhang C.

Auswertungen

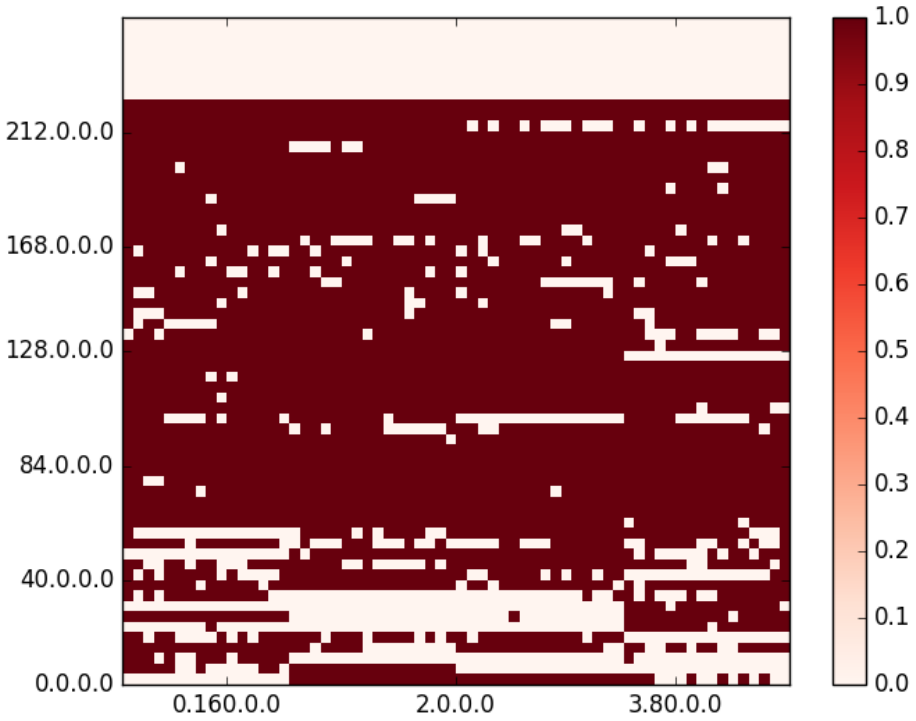


Abbildung C.1.: Quelladressen des gesamten empfangenen Datenverkehrs. Gruppiert nach /12 Netzen

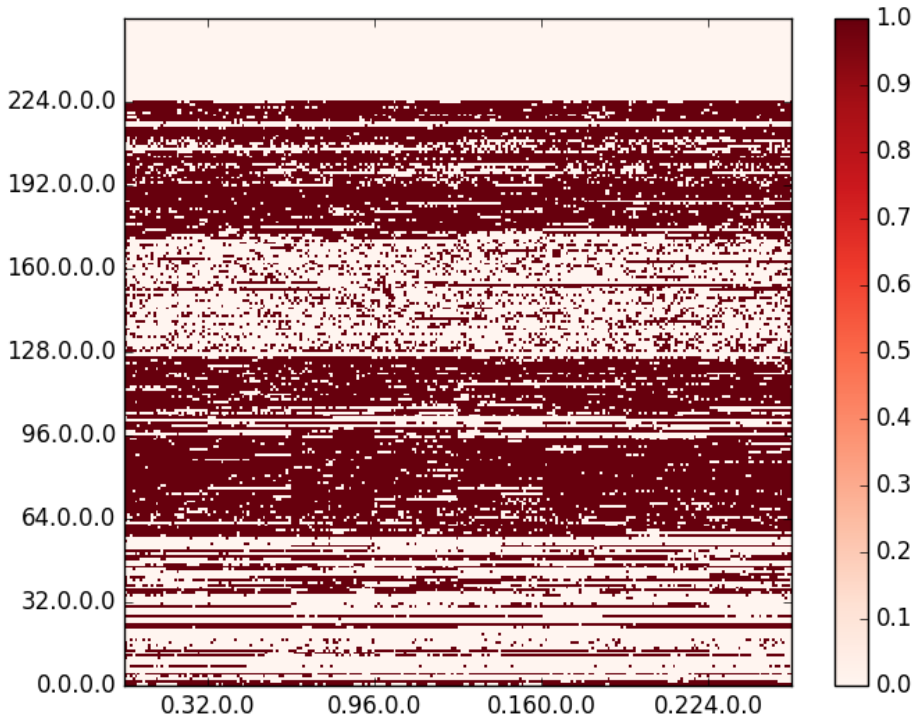


Abbildung C.2.: Quelladressen des gesamten empfangenen Datenverkehrs. Gruppiert nach /16 Netzen

Typ (Code)	Pakete	% Pakete	Bytes	% Bytes
DestinationUnreachable(CommAdminProhibited)	53.780	2,4	1.516.828	1,6
DestinationUnreachable(CommAdminProhibited)_AR	5722	0,3	269.604	0,3
DestinationUnreachable(FragmentationNeeded)	1107	0,0	61.364	0,1
DestinationUnreachable(FragmentationNeeded)_AR	71	0,0	3252	0,0
DestinationUnreachable(Host)	19.357	0,9	772.655	0,8
DestinationUnreachable(Host)_AR	393.664	17,5	16.303.238	17,3
DestinationUnreachable(HostAdminProhibited)	14.508	0,6	929.307	1,0
DestinationUnreachable(HostAdminProhibited)_AR	404.035	17,9	18.320.095	19,4
DestinationUnreachable(Net)	6921	0,3	697.423	0,7
DestinationUnreachable(Net)_AR	51.799	2,3	2.130.372	2,3
DestinationUnreachable(NetAdminProhibited)	17	0,0	868	0,0
DestinationUnreachable(NetAdminProhibited)_AR	70	0,0	2856	0,0
DestinationUnreachable(Port)	216.640	9,6	19.829.913	21,0
DestinationUnreachable(Port)_AR	12.692	0,6	574.068	0,6
DestinationUnreachable(Protocol)	1974	0,1	103.012	0,1
DestinationUnreachable(Protocol)_AR	131	0,0	6132	0,0
EchoReply	13.630	0,6	7.101.762	7,5
EchoRequest	748.340	33,2	12.760.370	13,5
EchoRequest(Code: 9)	136	0,0	16.320	0,0
TimeExceeded(FragmentReassemblyTimeExceeded)	11	0,0	6028	0,0
TimeExceeded(TTLExceeded)	91.356	4,0	3.442.766	3,7
TimeExceeded(TTLExceeded)_AR	219.912	9,7	9.386.484	10,0
TimestampReply	36	0,0	432	0,0

Tabelle C.1.: Aufschlüsselung der empfangenen ICMP-Pakete

Kennung	Verbindungen	% Verbindungen
2d5747304639302d(-WG0F90-)	239.218.995	91,8
2d6c74304339302d(-lt0C90-)	3.951.624	1,5
2d4445313335302d(-DE1350-)	3.746.975	1,4
2d4445313336302d(-DE1360-)	3.544.433	1,4
2d4d473231be302d(-MG21.0-)	2.199.022	0,8
2d4d47323110302d(-MG21.0-)	1.961.975	0,8
2d4d47323108302d(-MG21.0-)	962.680	0,4
2d4d47323114302d(-MG21.0-)	918.838	0,4
2d4d47323106302d(-MG21.0-)	437.827	0,2
2d6c74304432302d(-lt0D20-)	377.577	0,1
Gesamt()	260.670.432	98,7

Tabelle C.2.: Verwendete Clients im BitTorrent Datenverkehr

info_hash	Datei	% Verbindungen
6171805ce2ec402739b548e683bebb0968657993	wot_85.2472_85.2471_client.patch	25,6
8f228987e1af5e485f1a2158c4c80820d98052e8	wot_83.2100_client.patch	22,0
12c783f3071cbdf6feaec4f642eb3fc9d70c88cb	wot 82.1973 82.1972 client patch	12,1
bde86750bfdc16ee0e6b3d0662d9b920e484d86b	wot_85.2472_84.2364_client.patch	8,2
426f09550423e33d63ecb813c3bf11cf9d5e9516	wot_84.2250_83.2223_client.patch	5,5
46a9baf50af1915a461d59c6755eefffc5ddb726	wot_86.2661_client.patch	5,0
cdb063be28aae58ef3d2216bc049b1b95fe960cf	wot_82.1972_client.patch	4,6
11ae489156aaebbb985a4ec15cbcee3feabddaf8	wot_86.2662_86.2661_client.patch	3,7
fb4e58e19ade0aeb2bb4ea23da32d830713b59d6	wot_85.2471_client.patch	3,2
3515d33f496c9f6032c971e9b8589152e7143655	wot_81.1645_client.patch	2,3
9087a6ba5a567662c4c37e7dc7ee63e2d65e9877	wot 82.1973 81.1645	2,1
906b0283f9d5bd7c0cbf80b4e46e105880f38a44	Gorko 2013 Cam RIP	1,5
e61b8764c9ac4b0196acdb0516e9afd36b4330fb	wot_84.2249_client.patch	1,4
870557cbc23769f984e559736c4e3c2b9a53066f	wot_81.1645_80.1489_client.patch	1,2
99e9a8ba67ccc7201f0523a073a7d5222c994036	Djentelmeny	1,1
c9260fd0bb2aec2ff28ed0cb418fa04c31678f8c		0,3
0891132255ed95aab2cea5fcb1d82c28b802d4a2		0,1
d4db5aa919720f1518ca2d57f2f41617dc32eec6	Tek 082	0,0
9a114bae181c1c60ce09b05db7a3a974b31ee5c5	javset.com-midd-988.avi	0,0
9b255ed45e1bf6c928219b50bb0a7acd5c47a605	IPZ-630.avi	0,0
Summe		99,9

Tabelle C.3.: Häufigste `info_hash`es im BitTorrent Datenverkehr

Ländercode	Land	Verbindungen
RU	Russland	183.622.020
UA	Ukraine	31.476.915
BY	Belarus	22.953.436
KZ	Kazakhstan	6.475.053
DE	Deutschaldn	3.091.515
NL	Niedlerande	2.372.059
MD	Moldova	1.364.693
KG	Kyrgyzastan	1.301.654
AM	Armenien	869.982
PL	Polen	794.815

Tabelle C.4.: Länder, aus denen der BitTorrent Datenverkehr empfangen wurde

Methode	Verbindungen	% Verbindungen
GET	801.639	67,2
HEAD	151.913	12,7
nohttp	143.222	12,0
CONNECT	37.734	3,2
nopl	26.908	2,3
POST	24.594	2,1
OPTIONS	6018	0,5
XGET	86	0,0
SCANNER	8	0,0
LOCK	8	0,0
PROPFIND	2	0,0
PUT	1	0,0
Summe	1.192.133	100,0

Tabelle C.5.: Verwendete HTTP Methoden auf Port 80

Pfad	Verbindungen	% Verbindungen
/	285.896	28,0
/manager/html	22.414	2,2
/phpMyAdmin/scripts/setup.php	17.579	1,7
/pma/scripts/setup.php	17.190	1,7
/myadmin/scripts/setup.php	16.889	1,7
/rom-0	16.769	1,6
/phpTest/zologize/axa.php	16.550	1,6
http://www.google.com/	15.463	1,5
/w00tw00t.at.ISC.SANS.DFind:)	11.743	1,2
/muieblackcat	11.690	1,1
/tmUnblock.cgi	10.561	1,0
Summe	1.022.003	43,3

Tabelle C.6.: TOP 10 angefragte Ressourcen auf Port 80

Record Type	Verbindungen	% Verbindungen
Handshake	279.519	52,2
No TLS	242.471	45,3
No Payload	12.958	2,4
Alert	171	0,0
Change Cipher Spec	49	0,0
Application Data	40	0,0
Summe	535.208	100,0

Tabelle C.7.: Empfange TLS Record Types auf Port 443

Version	Verbindungen	% Verbindungen
TLS 1.0	249.109	89,0
TLS 1.1	15.036	5,4
SSL 3.0	9751	3,5
TLS 1.2	5701	2,0
unknown	182	0,1
Summe	279.779	100,0

Tabelle C.8.: Verwendete TLS Versionen auf Port 443

Typ	Verbindungen	% Verbindungen
client hello	279.459	99,98
unknown	40	0,01
hello request	17	0,01
certificate verify	1	0,00
server hello	1	0,00
client key exchange	1	0,00
Summe	279.519	100,00

Tabelle C.9.: Empfange TLS Handshake Typen auf Port 443

Typ	Anzahl	% Anzahl
no	254.044	90,79
DEFLATE	25.237	9,02
LZS	280	0,10
NULL	242	0,09
Summe	279.803	100,00

Tabelle C.10.: Kompressionsalgorithmen in *Client Hello* Paketen auf Port 443

Cipher	Anzahl	% Client Hello
TLS_RSA_WITH_AES_128_CBC_SHA	236.692	84,70
TLS_RSA_WITH_3DES_EDE_CBC_SHA	236.346	84,57
TLS_RSA_WITH_AES_256_CBC_SHA	235.043	84,11
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	224.877	80,47
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	224.446	80,31
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	223.642	80,03
TLS_RSA_WITH_RC4_128_SHA	222.720	79,70
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	220.461	78,89
TLS_ECDHE_RSA_WITH_RC4_128_SHA	205.397	73,50
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	203.534	72,83
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	199.924	71,54
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	159.895	57,22
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	155.433	55,62
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	149.688	53,56
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	149.533	53,51
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	139.788	50,02
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	138.432	49,54
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	137.055	49,04
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	135.723	48,57
TLS_RSA_WITH_RC4_128_MD5	132.195	47,30

Tabelle C.11.: 20 häufigste TLS *CipherSuites* in *Client Hello Paketen* auf Port 443

Typ	Pakete	Bytes
DNS	4.198.328	341.696.495
Netcore	2.358.240	110.893.475
SIP 5600	1.401.609	592.972.780
Fragment	17.366	228.252.381
BJNP	1.527.586	23.441.376
BitTorrent	1.355.606	90.133.668
SIP anderer Port	647.569	266.306.592
Summe	11.506.304	1.653.696.767
Gesamt	17.932.402	1.916.457.280
Prozent	64,16	86,29

Tabelle C.12.: Zusammenfassung des analysierten UDP Datenverkehrs

#	Merkmal	Zuordnung
1	SRC Port 53	DNS Backscatter
2	DST Port 5600	SIP Scanning
3	DST Port 53454	Netcore Exploit
4	DST Port 8000	Einzelphänomen
5	Payload Präfix `BJNP`	Canon Drucker
6	Payload Präfix `d1:ad2:i`	BitTorrent DHT
7	Payload Präfix `OPTIONS`	SIP Scanning (alternativer Port)

Tabelle C.13.: Iterationen für UDP-Datenverkehr

#	Merkmal	Zuordnung
1	SYN+ACK ohne SYN	Backscatter
2	Payload Präfix BitTorrent	BitTorrent
3	Scanner Fingerprint	Scanning
4	Scanverfahren (Stealth oder Port)	Scanning
5	kein Verbindungsaufbau	Artefakt
6	DST Port 20799, 48457	BitTorrent MSE
7	DST Port 22, 23, 3389, 5900	Scan Fernzugriff
8	DST Port 80, 443	HTTP(S) Zugriffe

Tabelle C.14.: Iterationen für TCP-Datenverkehr

Literatur

[APT07] Mark Allman, Vern Paxson und Jeff Terrell. „A brief history of scanning“. In: *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM. 2007, S. 77–82.

[Bai+05] Michael Bailey u. a. „The Internet Motion Sensor-A Distributed Blackhole Monitoring System.“ In: *NDSS*. 2005.

[Bai+06] Michael Bailey u. a. „Practical darknet measurement“. In: *Information Sciences and Systems, 2006 40th Annual Conference on*. IEEE. 2006, S. 1496–1501.

[BEP0003] Bram Cohen. *The BitTorrent Protocol Specification*. BEP 003. `http://bittorrent.org/beps/bep_0003.html`. BitTorrent.org, Jan. 2008. URL: `http://bittorrent.org/beps/bep_0003.html`.

[BEP0005] Arvid Norberg Andrew Loewenstern. *DHT Protocol*. BEP 005. `http://bittorrent.org/beps/bep_0005.html`. BitTorrent.org, Jan. 2008. URL: `http://bittorrent.org/beps/bep_0005.html`.

[BEP0011] The 8472. *Peer Exchange (PEX)*. BEP 011. `http://bittorrent.org/beps/bep_0011.html`. BitTorrent.org, Okt. 2015. URL: `http://bittorrent.org/beps/bep_0011.html`.

[Czy+13] Jakub Czyz u. a. „Understanding IPv6 internet background radiation“. In: *Proceedings of the 2013 conference on Internet measurement conference*. ACM. 2013, S. 105–118.

[DBH14] Zakir Durumeric, Michael Bailey und J Alex Halderman. „An internet-wide view of internet-wide scanning“. In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014, S. 65–78.

[DWH13] Zakir Durumeric, Eric Wustrow und J Alex Halderman. „ZMap: Fast Internet-wide scanning and its security applications“. In: *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. 2013, S. 605–620.

[GD12] Eduard Glatz und Xenofontas Dimitropoulos. „Classifying internet one-way traffic“. In: *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM. 2012, S. 37–50.

[Gra] Robert Graham. *MASSCAN: Mass IP port scanner*. `https://github.com/robertdavidgraham/masscan`. Accessed: 2017-03-22.

[HA05] Warren Harrop und Grenville Armitage. „Defining and evaluating greynets (sparse darknets)“. In: *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*. IEEE. 2005, S. 344–350.

[libtorrent] *Peer exchange should share only peers which finished handshake*. `https://code.google.com/archive/p/libtorrent/issues/407`. Accessed: 2016-01-12.

[Moo+04] David Moore u. a. *Network telescopes: Technical report*. Department of Computer Science und Engineering, University of California, San Diego, 2004.

[Moo+06] David Moore u. a. „Inferring internet denial-of-service activity“. In: *ACM Transactions on Computer Systems (TOCS)* 24.2 (2006), S. 115–139.

[Pan+04] Ruoming Pang u. a. „Characteristics of internet background radiation“. In: *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM. 2004, S. 27–40.

[pcap] *LibpcapFileFormat*. `https://wiki.wireshark.org/Development/LibpcapFileFormat`. Accessed: 2017-02-22.

[Pro] Niels Provos. *Honeyd Virtual Honeypot*. `http://www.honeyd.org/`. Accessed: 2017-03-28.

[RFC0791] Jon Postel. *Internet Protocol*. STD 5. `http://www.rfc-editor.org/rfc/rfc791.txt`. RFC Editor, Sep. 1981. URL: `http://www.rfc-editor.org/rfc/rfc791.txt`.

[RFC0792] J. Postel. *Internet Control Message Protocol*. STD 5. `http://www.rfc-editor.org/rfc/rfc792.txt`. RFC Editor, Sep. 1981. URL: `http://www.rfc-editor.org/rfc/rfc792.txt`.

[RFC0793a] J. Postel. *User Datagram Protocol*. STD 6. `http://www.rfc-editor.org/rfc/rfc768.txt`. RFC Editor, Aug. 1980. URL: `http://www.rfc-editor.org/rfc/rfc768.txt`.

[RFC0793b] Jon Postel. *Transmission Control Protocol*. STD 7. http://www.rfc-editor.org/rfc/rfc793.txt. RFC Editor, Sep. 1981. URL: http://www.rfc-editor.org/rfc/rfc793.txt.

[RFC0854] J. Postel und J. Reynolds. *Telnet Protocol Specification*. STD 8. http://www.rfc-editor.org/rfc/rfc854.txt. RFC Editor, Mai 1983. URL: http://www.rfc-editor.org/rfc/rfc854.txt.

[RFC1035] P. Mockapetris. *Domain names - implementation and specification*. STD 13. http://www.rfc-editor.org/rfc/rfc1035.txt. RFC Editor, Nov. 1987. URL: http://www.rfc-editor.org/rfc/rfc1035.txt.

[RFC1945] Tim Berners-Lee, Roy T. Fielding und Henrik Frystyk Nielsen. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. http://www.rfc-editor.org/rfc/rfc1945.txt. RFC Editor, Mai 1996. URL: http://www.rfc-editor.org/rfc/rfc1945.txt.

[RFC2246] Tim Dierks und Christopher Allen. *The TLS Protocol Version 1.0*. RFC 2246. http://www.rfc-editor.org/rfc/rfc2246.txt. RFC Editor, Jan. 1999. URL: http://www.rfc-editor.org/rfc/rfc2246.txt.

[RFC2577] M. Allman und S. Ostermann. *FTP Security Considerations*. RFC 2577. RFC Editor, Mai 1999.

[RFC2616] Roy T. Fielding u. a. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. http://www.rfc-editor.org/rfc/rfc2616.txt. RFC Editor, Juni 1999. URL: http://www.rfc-editor.org/rfc/rfc2616.txt.

[RFC3261] J. Rosenberg u. a. *SIP: Session Initiation Protocol*. RFC 3261. http://www.rfc-editor.org/rfc/rfc3261.txt. RFC Editor, Juni 2002. URL: http://www.rfc-editor.org/rfc/rfc3261.txt.

[RFC5246] T. Dierks und E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. http://www.rfc-editor.org/rfc/rfc5246.txt. RFC Editor, Aug. 2008. URL: http://www.rfc-editor.org/rfc/rfc5246.txt.

[Sch13] Patrick Schumacher. „Analyse und Klassifikation von Blackhole-Internetverkehr. Bachelorarbeit“. Universität Duisburg-Essen, Sep. 2013.

[SKR08] Sohraab Soltani, Syed A Khayam und Hayder Radha. „Detecting malware outbreaks using a statistical model of blackhole traffic“. In: *2008 IEEE International Conference on Communications*. IEEE. 2008, S. 1593–1597.

[Tan98] Andrew S. Tanenbaum. *Ccomputernetzwerke*. 3., revidierte Auflage. Prentice Hall, 1998.

[Var+16] Matteo Varvello u. a. „Is the Web HTTP/2 Yet?“ In: *International Conference on Passive and Active Network Measurement*. Springer. 2016, S. 218–232.

[Wus+10] Eric Wustrow u. a. „Internet background radiation revisited“. In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM. 2010, S. 62–74.

[YBP04] Vinod Yegneswaran, Paul Barford und Dave Plonka. „On the design and use of Internet sinks for network abuse monitoring“. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2004, S. 146–165.

[Zim80] Hubert Zimmermann. „OSI reference model–The ISO model of architecture for open systems interconnection“. In: *IEEE Transactions on communications* 28.4 (1980), S. 425–432.