FLORIAN E. W. ADAMSKY

STATISTISCHE ANALYSE ZUR IDENTIFIZIERUNG UND KLASSIFIZIERUNG VON PROTOKOLLEN IN IP-STRÖMEN

STATISTISCHE ANALYSE ZUR IDENTIFIZIERUNG UND KLASSIFIZIERUNG VON PROTOKOLLEN IN IP-STRÖMEN

FLORIAN E. W. ADAMSKY

zur Erlangung des akademischen Grades
Bachelor of Science

vorgelegt dem
Fachbereich Informationstechnik, Elektrotechnik und Mechatronik
der Fachhochschule Gießen-Friedberg

Juni 2010

Florian E. W. Adamsky: *Statistische Analyse zur Identifizierung und Klassifizierung von Protokollen in IP-Strömen,* Bachelor of Science, © Juni 2010

REFERENT:
Prof. Dr. rer. nat. Rudolf Jäger

KOREFERENT:
Dipl. Ing. (FH) Christopher Köhnen

ZUSAMMENFASSUNG

Die vorliegende Bachelor-Thesis zeigt Techniken auf, mit denen man Netzwerk-Protokolle identifizieren kann, die in Schicht 7 des OSI-Modells angesiedelt sind, und weist auf ihre Einsatzgebiete hin. Der Fokus der Arbeit liegt auf dem SPID-Algorithmus, den Hjelmvik et al. entwickelte. Die Technik versteht sich als Hybrid-Form und verwendet Elemente aus der *Deep-Packet-Inspektion* und behandelt diese wie statistische Werte. Neben einer detaillierten Darstellung des Algorithmus, wird eine eigene Implementierung mit Verbesserungen vorgestellt.

Alle Techniken werden nach den folgenden Anforderungen von *QoSiLAN* – einer Quality-of-Service Technik – abgewogen: Erstens sollen Protokolle in naher Echtzeit erkannt werden, um nachfolgend einen IP-Strom zu priorisieren. Zweitens soll eine möglichst robuste und sichere Identifizierung von Streaming-Protokollen möglich sein. Und Drittens soll die Technik auf leistungsarmer und kostengünstiger Hardware laufen. Der SPID-Algorithmus wird hinsichtlich der aufgeführten Anforderungen evaluiert und die gewonnenen Ergebnisse dargestellt und kritisch bewertet.

VERZEICHNIS DER VERÖFFENTLICHUNGEN

Einige Ideen und Abbildungen sind vorher in den folgenden wissenschaftlichen Publikationen erschienen:

Köhnen, Christian ; Überall, Christian ; Adamsky, Florian ; Rakocevic, Veselin ; Rajarajan, Muttukrishnan ; Jäger, Rudolf : Enhancements to Statistical Protocol IDentification (SPID) for Self-Organised QoS in LANs. In: *ICCCN 2010 Track on Network Algorithms, Performance Evaluation and Theory (NAPET) (ICCCN 2010 NAPET).* Zurich, Switzerland, 8 2010

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [Knu74]

INHALTSVERZEICHNIS

1 EINLEITUNG 1
1.1 Ziel der Arbeit . . . 2
1.2 Aufbau der Arbeit . . . 3
2 NETZWERKPROTOKOLL IDENTIFIZIERUNG 5
2.1 Begriffsdefinition . . . 5
2.2 Einsatzmöglichkeiten . . . 6
2.2.1 Netzwerk Design . . . 6
2.2.2 Politische und soziale Kontrolle . . . 9
2.3 Techniken . . . 10
2.3.1 TCP/UDP Portnummern . . . 10
2.3.2 Deep Packet Inspection . . . 11
2.3.3 Maschinelles Lernen . . . 12
3 STATISTICAL PROTOCOL IDENTIFICATION 15
3.1 Algorithmus Details . . . 15
3.1.1 Gruppierung der Datenpakete in Flows . . . 16
3.1.2 Approximierung mit dem empirischen Gesetz der großen Zahlen . . . 18
3.1.3 Identifizierung mittels Kullback-Leibler Divergenz . . . 19
3.2 Implementierung . . . 20
3.2.1 Neue Leistungsmerkmale . . . 23
3.2.2 Messverfahren zur Ermittlung der Protokolleigenschaften . . . 24
3.2.3 Darstellung der Fingerprint-Datenbank . . . 34
4 EVALUATION DES ALGORITHMUS 37
4.1 Datenbestand . . . 37
4.1.1 Beschaffung aus frei verfügbaren Quellen . . . 37
4.1.2 Erstellung von Mitschnitten . . . 37
4.2 Evaluationsmethoden . . . 41
4.3 Ergebnisse . . . 43
4.3.1 Ergebnisse in Abhängigkeit der approximierten IP-Ströme . . . 44
4.3.2 Ergebnisse in Abhängigkeit der untersuchten Pakete . . . 44
4.3.3 Ergebnisse in Abhängigkeit des Grenzwertes 45
4.3.4 Speicherverbrauch und Geschwindigkeitstest 45
5 FAZIT 53
A APPENDIX 55

LITERATURVERZEICHNIS 57

ABBILDUNGSVERZEICHNIS

Abbildung 1 Beispiel für die Identifizierung von Netzwerkprotokollen . . . 2

Abbildung 2 Die Schichten des TCP/IP-Referenzmodells 5

Abbildung 3 Die drei Phasen des SPID-Algorithmus . . . 15
Abbildung 4 Aufbau eines TCP-Pakets . . . 16
Abbildung 5 Aufbau und Abbau einer TCP-Verbindung . 17
Abbildung 6 Visualisierung des Gesetzes der großen Zahlen . . . 19
Abbildung 7 Klassendiagramm der SPID-Implementierung 21
Abbildung 8 Sequenzdiagramm der SPID-Implementierung 23
Abbildung 9 Vergleich der Byte-Frequenz . . . 25
Abbildung 10 Durchschnittliche Anzahl der Richtungswechsel . . . 27
Abbildung 11 Durchschnittliche prozentuale Datenmenge pro Richtung . . . 28
Abbildung 12 Durchschnittliche Entropie beim ersten Paket in jede Richtung . . . 29
Abbildung 13 Ausschnitt aus Wireshark . . . 31
Abbildung 14 Durchschnittliche Paketgröße des ersten Pakets . . . 33

Abbildung 15 Evaluation des Algorithmus . . . 42
Abbildung 16 Ergebnisse der Protokoll-Identifizierung . . 43
Abbildung 17 Abhängigkeit der approximierten Flows . . 44
Abbildung 18 Abhängigkeit der untersuchten Pakete . . . 45
Abbildung 19 Abhängigkeit des Grenzwertes . . . 46
Abbildung 20 Linksys WRT54GL mit SD-Karten Modifikation und IBM Thinkpad T40p . . . 46
Abbildung 21 Versuchsaufbau des Geschwindigkeitstest . 48
Abbildung 22 Leistungstest in Abhängigkeit der Paketgröße bei dem IBM Thinkpad T40p . . . 49
Abbildung 23 Leistungstest in Abhängigkeit der Paketgröße beim Linksys WRT54GL . . . 50
Abbildung 24 Leistungstest in Abhängigkeit der Anzahl der Protokolle in der Datenbank . . . 50

TABELLENVERZEICHNIS

Tabelle 1 Markante Bytes bei HTTP 26
Tabelle 2 Exemplarische Darstellung von aufeinanderfolgenden Bytes 32
Tabelle 3 Beispiel der Fingerprint-Datenbank für HTTP 35
Tabelle 4 Konfusionsmatrix für binäre Ergebnisse . . . 42
Tabelle 5 Validierungs-Ergebnisse 55

AUFLISTUNGSVERZEICHNIS

Auflistung 1 Beispiel einer Template-Methode von einem Messverfahren 22
Auflistung 2 Teil eines HTTP-Headers in Byte- und ASCII-Form 24
Auflistung 3 Beispiel der Quersumme bei den ersten 4 Bytes von HTTP 30
Auflistung 4 Ausgabe des Programms pmap auf dem Linksys WRT54GL 47
Auflistung 5 Modifikation um die Zeit zu messen 49

ABKÜRZUNGSVERZEICHNIS

ACK	Acknowledgment
ASCII	American Standard Code for Information Interchange
BT	British Telecom
CDDBP	CD Database Protocol
DPI	Deep Packet Inspection

FIN	Finish
FN	False Negative
FP	False Positve
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IDPS	Intrusion Detection Prevetion Systems
IDS	Intrusion Detection System
IGMP	Internet Group Management Protocol
IM	Instant Messaging
IPS	Intrusion Prevention System
IPTV	Internet Protocol Television
IPsec	IP security
IP	Internet Protocol
IRC	Internet Relay Chat
ISP	Internet Service Provider
KLD	Kullback-Leibler Divergenz
LAN	Local Area Network
LLTD	Link Layer Topology Discovery
MMS	Microsoft Media Server Protocol
MPEG-TS	Moving Picture Experts Group Transport Stream
MSE	Message Stream Encryption
MTU	Maximal Transmission Unit
NSA	National Security Agency
NSLP	NSIS Signaling Layer Protocol
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
POP3	Post Office Protocol Version 3
POSIX	Portable Operating System Interface

QoE	Quality of Experience
QoSiLAN	Quality of Service for Local Area Networks
QoS	Quality of Service
RAID	Redundant Array of Independent Disks
RST	Reset
RTMP	Real Time Messaging Protocol
RTP	Real-Time Transport Protocol
SMTP	Simple Mail Transfer Protocol
SOHO	Small Office, Home Office
SPID	Statistical Protocol Identification
SSH	Secure Shell
SSL	Secure Sockets Layer
SYN	Synchronize
TCP	Transmission Control Protocol
Telnet	Telecommunication Network
TFTP	Trivial File Transfer Protocol
TP	True Positve
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VoIP	Voice over Internet Protocol
WMV	Windows Media Video
WWW	World Wide Web

1 EINLEITUNG

Das Internet muss heutzutage höheren Belastungen standhalten, als noch zu Zeiten seiner Entstehung. Dafür ist teilweise die zunehmende Anzahl an Benutzern verantwortlich, die Dienste wie Internet-Fernsehen (IPTV) und -Telefonie (VoIP) verwendet. Der Branchenverband Bitkom erwartet für das Jahr 2010 einen Zuwachs von annähernd 1,8 Millionen Nutzern von IPTV-Angeboten (vgl. [Buno9a]); bei der Internet-Telefonie sagt der Verband einen Zuwachs von 21 % voraus (vgl. [Buno9b]). Die Onlinevideo-Plattform *YouTube* meldete Anfang Oktober 2009, dass mehr als eine Milliarde Videoabrufe pro Tag verzeichnet werden (vgl. [Huro9]). Die Akzeptanz und damit der Erfolg solcher Dienste hängt von der Qualität der Verbindung ab. Verzögerungen oder Überlastungen bestraft der Anwender dadurch, dass er auf eines der vielen anderen Angebote ausweicht. Um die Bandbreite für diese Dienste zu garantieren, gibt es *Quality of Service (QoS)*-Strategien wie *IntServ* oder *DiffServ*, die das Netzwerk unterstützen muss. Viele Haushalte sind mit Hardware ausgestattet, die diese Techniken nicht integriert haben. Zusätzlich ist es für Anbieter von bandbreitenintensiven Dienstleistungen nicht möglich, das Netzwerk des Kunden einzusehen, so dass der Anbieter keine geeigneten Maßnahmen treffen kann. Eine Einflussnahme ist somit lediglich bis zum Anschlußpunkt des Hauses möglich. Das Nicht-Anwenden von QoS-Strategien führt zu einer geringeren Annahme des Dienstes und das wiederum zu einem niedrigeren *Quality of Experience (QoE)* ([Kilo8]).

Ein Ansatz um den QoE zu erhöhen ist *Quality of Service for Local Area Networks (QoSiLAN)*. Die Technik arbeitet in drei Phasen: Die erste Phase analysiert das Netzwerk und erstellt einen detaillierten Plan über die physikalisch angebundenen Komponenten in einem Netzwerk und prüft die zur Verfügung stehenden Bandbreite-Ressourcen. Für diesen Zweck wurde das Protokoll *Link Layer Topology Discovery (LLTD)* neu implementiert und erweitert. Die zweite Phase analysiert den Internet-Verkehr und erkennt Protokolle. Die letzte Phase von QoSiLAN reserviert und priorisiert die Bandbreite. Hierzu wird das signalbasierte Protokoll *NSIS Signaling Layer Protocol (NSLP)* für QoS genutzt, um die Knappheit der Bandbreite mit den verschiedenen Geräte auszutauschen. Im Vergleich zu anderen QoS-Lösungen, ist QoSiLAN nicht abhängig von den QoS-Funktionalitäten der Router und Switches, kann diese aber bei Bestehen nutzen (vgl. [Köho9, S. 29–31]).

1.1 ZIEL DER ARBEIT

Die Untersuchung der zweiten Phase ist Gegenstand der vorliegenden Bachelor-Thesis. Die Abbildung 1 exemplifiziert die Netzwerkprotokoll-Identifizierung, anhand eines typischen Heimnetzwerks.

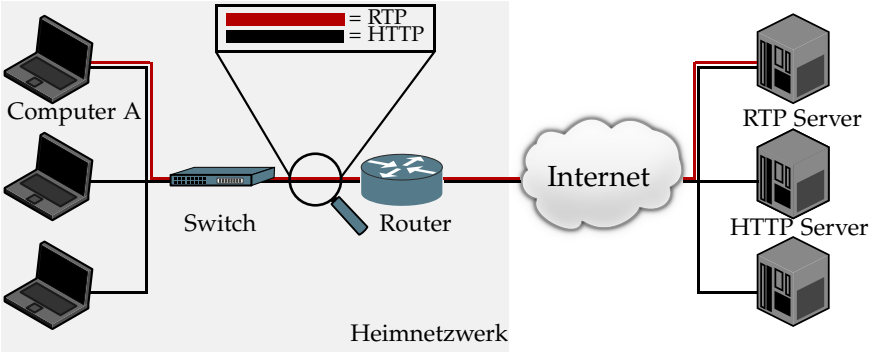

Abbildung 1: Beispiel für die Identifizierung von Netzwerkprotokollen. Quelle: eigene Darstellung auf Basis von [Wiko7a].

Der Benutzer von Computer A surft auf Webseiten und schaut parallel dazu einen Film, der über das Streaming-Protokoll *Real-Time Transport Protocol (RTP)* empfangen wird. Die Netzwerkprotokoll-Identifizierung ist in Abbildung 1 mit Hilfe einer Lupe dargestellt und erkennt die Netzwerkprotokolle durch das Analysieren der vorbeifließenden Daten.

QoSiLAN stellt an die Identifizierung drei grundlegende Anforderungen:

NAHE ECHTZEIT-ERKENNUNG: Die Identifizierung von Protokollen muss schnellst-möglich erfolgen, damit das Protokoll noch priorisiert werden kann.

STREAMING-PROTOKOLLE: Das Hauptaugenmerk liegt auf der Erkennung von Streaming-Protokollen oder solchen Protokollen, die es wert sind, priorisiert zu werden.

PORTABILITÄT: Die Erkennung soll möglichst unabhängig von der Hardware und dem Betriebssystem sein, so dass auch *Small Office, Home Office (SOHO)*-Geräte für den Einsatz in Betracht kommen.

In dieser Bachelor-Thesis werden verschiedene Algorithmen vorgestellt und geprüft, ob sich mit diesen vor allem Streaming-Protokolle in naher Echtzeit erkennen lassen. Der Schwerpunkt bildet der *Statistical Protocol Identification (SPID)*-Algorithmus, der neu programmiert und so auf unterschiedliche Art und Weise verbessert wurde. Weiterhin werden anhand mehrerer Beispiele

zusätzliche Einsatzgebiete skizziert, die sich mit einer Identifizierung von Netzwerk-Protokollen erschließen. Abschließend wurde der SPID-Algorithmus nach verschiedenen Kriterien evaluiert und geprüft, ob dieser geeignet ist, um die zweite Phase von QoSiLAN zu übernehmen.

1.2 AUFBAU DER ARBEIT

Der Aufbau der Arbeit gliedert sich wie folgt: Nach der Einleitung, in der Thema und Zugang dargestellt werden, werden im zweiten Kapitel die verwendeten Begriffe definiert und so das Fundament für den Gegenstand der Arbeit gelegt. Weiterhin werden Techniken zur Identifizierung von Netzwerk-Protokollen analysiert und nach den oben aufgeführten Anforderungen bewertet. Das dritte Kapitel stellt den SPID-Algorithmus dar und geht auf die Implementierung detailliert ein. Im letzten Kapitel wird der entwickelte Algorithmus hinsichtlich der Anforderungen evaluiert und die gewonnenen Ergebnisse dargestellt und kritisch bewertet.

2 NETZWERKPROTOKOLL IDENTIFIZIERUNG

In diesem Kapitel werden die Begriffe definiert, die essenziell sind, für das Verständnis der inhaltlichen Arbeit. Weiterhin werden die Einsatzgebiete der Netzwerkprotokoll-Identifizierung skizziert und verschiedene Anwendungsbeispiele aufgezeigt. Im letzten Unterkapitel werden die Techniken erläutert, die genutzt werden können, um Netzwerkprotokoll zu identifizieren und nach ihren Vor- und Nachteilen abgewägt. Dieses Kapitel bildet das Fundament für den Gegenstand der vorliegenden Arbeit.

2.1 BEGRIFFSDEFINITION

Unter dem Begriff *Netzwerkprotokoll* wird eine exakte Konvention von Regeln und Formaten verstanden, die die Kommunikation zwischen Teilnehmern in einem Netzwerk regelt. Die Aufgaben der Protokolle unterteilen sich in Schichten oder Ebenen (engl. layers), um die Komplexität zu reduzieren (vgl. [Tan03, S. 42]). In Abbildung 2 ist das TCP/IP-Referenzmodell zu sehen, welches für die Internetprotokollfamilie maßgeblich ist, für die Einteilung der Schichten (vgl. [Ste93, S. 2]).

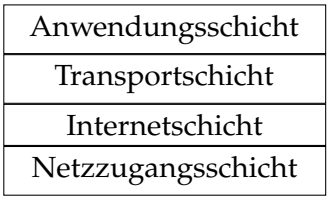

Abbildung 2: Die Schichten des TCP/IP-Referenzmodells. Quelle: eigene Darstellung.

Die Aufgaben der Schichten lassen sich wie folgt definieren (vgl. [Ste93, S. 2]):

NETZZUGANGSSCHICHT: Diese Schicht ist verantwortlich für die Technik der Datenübertragung, die in einem Netzwerk eingesetzt wird. Dies umfasst die ganze Hardware, die notwendig ist, um Daten zu übertragen, einschließlich der Treiber des Betriebssystems und dem Übertragungsmedium.

INTERNETSCHICHT: Die Internetschicht hat die Aufgabe, Datenpakete an den richtigen Empfänger zuzustellen. Hierunter fällt auch das Routing der Pakete und die Adressierung der

Teilnehmer. Zu dieser Schicht zählen die folgenden Protokolle: *Internet Protocol (IP)*, *Internet Control Message Protocol (ICMP)* und *Internet Group Management Protocol (IGMP)*.

TRANSPORTSCHICHT: Die Transportschicht definiert die Details der Übertragung und lässt sich in zwei Hauptprotokolle aufteilen: *Transmission Control Protocol (TCP)* ist ein verbindungsorientiertes und zuverlässiges Protokoll, dass die Übertragung sicherstellt. Im Unterschied zu TCP ist das *User Datagram Protocol (UDP)* ein verbindungsloses und unzuverlässiges Protokoll, das keinen Garant für den Empfang gibt.

ANWENDUNGSSCHICHT: Die Anwendungsschicht umfasst alle Programme, die die unteren Schichten für die Datenübertragung nutzen. Anwendungen für diese Schicht sind beispielsweise *Hypertext Transfer Protocol (HTTP)*, *File Transfer Protocol (FTP)* oder E-Mail.

Der Begriff *Netzwerkprotokoll-Identifizierung* bezieht sich in der vorliegenden Arbeit auf die Anwendungsschicht. Der Grund hierfür ist, dass die Protokolle der Netzzugangs- und Internetschicht bekannt sein müssen, damit eine Übertragung stattfinden kann. Bei der Transportschicht steht das verwendete Protokoll im IP-Kopf (engl. Header) im Feld *Protocol*. Daher ist eine Identifizierung nur innerhalb der Anwendungsschicht sinnvoll. Im *Open Systems Interconnection (OSI)*-Referenzmodell ist die Anwendungsschicht nochmal in die Anwendungs-, Darstellungs- und Kommunikationssteuerungsschicht unterteilt. Weitere übliche Begriffe für diese Technik sind *Application Identification*, *Port Independent Protocol Identification*, *Protocol Discovery*, *Application Recognition* und *Traffic Classification* (vgl. [Hje09]).

2.2 EINSATZMÖGLICHKEITEN

2.2.1 *Netzwerk Design*

Das Netzwerk-Design umfasst alles von der Planung bis zum Aufbau eines Netzwerkes. Hierzu zählen auch die Bereiche Sicherheit und Bandbreiten-Management, die besonders bei der Identifizierung von Protokollen eine Rolle spielen.

2.2.1.1 *Netzwerk-Sicherheit*

Ein möglicher Einsatz der Identifizierung von Programmen oder Protokollen ist die Netzwerk-Sicherheit. Dieses weit gefächerte Gebiet, umfasst die Probleme mit Schadsoftware (engl. malware), Viren, Würmer, trojanische Pferde, Einbrüche in Computer-

systeme und vieles mehr. Diesen Problemen versucht man mit verschiedenen Techniken entgegen zu wirken.

Zwei populäre Vertreter sind das *Intrusion Detection System (IDS)* und *Intrusion Prevention System (IPS)*. Das IDS ist ein Verfahren, dessen Hauptaufgabe darin besteht, den Computer oder das Netzwerk zu überwachen und Angriffe zu detektieren. Nachdem das IDS einen Angriff erkannt hat, versucht das IPS diesen abzuwehren (vgl. [SM07, S. 2-1]). *Intrusion Detection Prevetion Systems (IDPS)* ist eine Symbiose aus beiden Techniken. Die Protokoll-Identifizierung unterstützt vor allem die Netzwerk-Überwachung, um verdächtige Programme oder Angriffe zu erkennen.

In den Bereich Netzwerk-Sicherheit fällt auch das Thema Firewall. Firewalls schützen das Intranet oder *Local Area Network (LAN)* vor nicht autorisiertem Zugriff aus dem Internet (vgl. [ZCC00, S. 21]). Der Schutz erfolgt durch Sicherheitsrichtlinien, die der Administrator zuvor definiert. Eine Technik die dieses Konzept umsetzt, nennt sich *packet filtering* (vgl. [ZCC00, S. 165]). Hierbei wird jedes Paket auf Kriterien geprüft und anschließend entschieden, ob das Paket akzeptiert, abgelehnt oder verworfen wird. Die Prüfung fokussiert sich auf den *header* des Pakets. Mit der Netzwerkprotokoll-Identifizierung können anwendungsspezifische Richtlinien definiert werden, wie etwa das Verwerfen aller Telnet-Verbindungen oder die Akzeptanz des Dienstes *Secure Shell (SSH)* auf dem Port 22. Hierdurch kann die Firewall die Sicherheitsrichtlinien besser umzusetzen.

2.2.1.2 *Bandbreiten-Management mit QoS*

Ein wichtiges Kriterium im Netzwerk-Design ist die Verteilung der zur Verfügung stehenden Bandbreite. Die vier primären Anforderungen an einen Datenfluss sind laut Tanenbaum [Tan03, S. 437]: Zuverlässigkeit, Übertragungsverzögerung, Jitter und Bandbreite. Diese Parameter beschreiben die Qualität des Datenflusses und werden als Dienstgüte oder QoS bezeichnet. Zeitkritische Dienste wie *Internet Protocol Television* (IPTV) oder *Voice over Internet Protocol* (VoIP) haben hohe Anforderungen an kurze Verzögerungszeiten (engl. delay). Bei *Video-on-Demand* und *File-Sharing* kommt zusätzlicher Bedarf an hoher Bandbreite hinzu. Besonders empfindlich ist die durchschnittliche Laufzeit der Pakete, die als *Jitter* bezeichnet wird. Ob ein Paket exakt zwei Sekunden bis zum Empfänger benötigt, ist unerheblich, solange die Zeit konstant bleibt. Variiert der *Jitter* stark zwischen den Paketen, verschlechtert sich die Qualität. Router sollten diesen Diensten eine höhere Priorität einräumen, als solchen, die weniger zeitkritisch sind, wie etwa E-Mail, Web-Surfen oder eine Chat-Anwendung. Die Identifizierung ist auch hier Essenz, um eine Priorisierung vorzunehmen. Das beschriebene Verfahren bezieht sich auf Datennetze,

in denen ein Internet-Anbieter eine Leitung zur Verfügung stellt. Der Anbieter kann diese Techniken allerdings auch selbst nutzen um beispielsweise die eigenen Infrastruktur zu entlasten oder dem Kunden neue Datentarife anzubieten.

2.2.1.3 *Graduelle Datentarife und Ad-Injection*

Der Netzbetreiber kann Anwendungen drosseln oder sperren, die zu viel Bandbreite belegen. Dadurch schützt der Betreiber sich vor Überbuchung (engl. overbooking) und entlastet die eigene Infrastruktur. Beispielgebend war der US-Kabelnetzbetreiber *Comcast*, der 2007 die *Peer-to-Peer (P2P)*-Software BitTorrent gedrosselt und teilweise gesperrt hatte (vgl. [Kre07]). Daraufhin entwickelte sich eine öffentliche Debatte um das Thema *Netzneutralität* (vgl. [Kre06]), welches die Frage aufwirft, ob und inwieweit es dem *Internet Service Provider (ISP)* erlaubt sein soll, spezifische Dienste auszubremsen.

Ein weiteres Modell ist, die gedrosselten oder gesperrten Leitungen gegen Aufpreis dem Kunden als zusätzliche Option anzubieten. In Deutschland erklärte im Jahr 2006 der damalige Telekom-Chef Kai-Uwe Ricke in einem Interview der Wirtschaftswoche, dass er Webriesen wie Google, Yahoo, Amazon und Ebay extra zur Kassen bitten möchte (vgl. [Han06]). Auch im Jahr 2010 fordert der Telekom-Chef Obermann, dass Anbieter von datenintensiven Inhalten zusätzliche Gebühren entrichten müssen (vgl. [Hag10]).

Denkbar ist auch, dass der Betreiber gezielt Werbung in den Datenverkehr injiziert und sich somit dem Verfahren *Ad-Injection* bedient. Populär geworden ist damit die Firma *Phorm*, die mit der Software *WebWise* Datenverkehr analysiert und personalisierte Werbung einschleust. Der Internet-Provider *British Telecom (BT)* hat die Software heimlich bei rund 18.000 Kunden angewendet und dafür viel negative Kritik geerntet. Daraufhin gab der Provider bekannt, dass vorerst auf den Einsatz der Software verzichtet wird (vgl. [Fis08] und [Bri09]). Das Verfahren ermöglicht es, günstige Datentarife mit Werbung und teurere Tarife ohne Werbung anzubieten.

Erst durch die sichere Identifizierung der Anwendung sind solche graduellen – in Stufen eingeteilte – Datentarife möglich. Netzaktivisten wie der Erfinder des *World Wide Web (WWW)* Sir Tim Berners-Lee warnen vor solchen Eingriffen, da dadurch das Ende-zu-Ende-Prinzip (engl. end-to-end principle)[1] verletzt wird und sich so eine Zweiklassen-Gesellschaft im Internet etabliert (vgl. [Fil06]).

1 Wesentliches Internet-Grundprinzip, das besagt, dass die Intelligenz nur an den End-Punkten stattfinden soll und nicht bei der Übertragung (vgl. [HL93, S. 269]

2.2.2 *Politische und soziale Kontrolle*

Vertreter aus Wirtschaft und Politik präsentieren Techniken oft als potemkinsche Dörfer, die dazu dienten soziale und politische Problemen zu lösen, wie etwa: Videoüberwachung bei Kriminalität oder Kopiersperren bei Urheberrechtsverletzungen. Die Technik zur Identifizierung von Protokollen bildet dabei keine Ausnahme, vor allem weil das Internet ein Spiegel der Gesellschaft ist und die Probleme dort deutlich sichtbarer werden.

2.2.2.1 *Abhören und Überwachen*

Wie das Thema *Netzneutralität* ist auch die Nutzung der Technik zum Abhören und Überwachen des Datenverkehrs sehr kontrovers, da vehement in die Privatsphäre der Menschen eingegriffen wird. Durch die Erkennung der Anwendung kann der Geheimdienst oder eine Strafverfolgungsbehörde den Datenverkehr gezielt mitschneiden und auswerten. In den Vereinigten Staaten von Amerika hat im Dezember 2005 ein Informant bekannt gegeben, dass der Internet-Provider AT&T auf Anweisung der *National Security Agency (NSA)*, Hardware eingebaut hat, um Datenpakete zu kopieren und auszuwerten (vgl. [Ben09, S. 24]). Ein Jahr später bestätigte der AT&T-Techniker Mark Klein die Vorwürfe (vgl. [Kle06]).

2.2.2.2 *Filtern und Zensieren*

Mit dieser Technik lassen sich unliebsame Inhalte filtern und zensieren. Ein Beispiel ist Jugendschutz-Software, die die Eltern auf dem heimischen Computer installieren können. Die Software soll Kinder und Jugendliche vor gewalttätigen oder pornographischen Inhalten im Internet schützen, indem sie jede Internet-Aktivität nach Schlagwörtern durchsucht. Für den Fall, dass die Software fündig wird, gibt es zwei Vorgehensweisen: Das *blacklist*- und *whitelist*-Verfahren. Wenn die Software ein Schlagwort findet, wird sie den Zugang bei dem *blacklist*-Verfahren verwehren und bei dem *whitelist*-Verfahren erlauben.

Die Filter dienen nicht nur dem Schutz der Minderjährigen, sondern auch der Kontrolle einer Bevölkerung. Regierungen können diese Kontrolle ausüben, indem sie Inhalte zensieren wie etwa: Regierungskritik, Pornographie, Religion, Glücksspiel oder Urheberrechtsverletzungen. Federführend sind Länder wie China, Iran oder Tunesien. Laut Angaben eines Berichts der Organisation *Reporter ohne Grenzen* zensierten im Jahr 2009 rund 60 Länder Inhalte im Internet – und die Tendenz ist steigend (vgl. [MJ10, S. 2]). Selbst Deutschland ist vor dieser Kontrolle nicht gefeit. Im November 2008 schlug die Bundesfamilienministerin Ursula von der Leyen vor, Filtertechniken einzusetzen, um die Verbreitung

von dem dokumentieren Kindesmissbrauch zu erschweren (vgl. [GR08]). Der Vorschlag stieß auf massive Kritik von Seiten der Netzaktivisten, Bürgerrechtlern, Missbrauchsopfern und wurde als *Zensursula-Debatte* bekannt. Der Vorschlag ist in Form eines Gesetzes mit dem Namen *Zugangserschwerungsgesetz* am 18. Juni 2009 vom Bundestag beschlossen worden (vgl. [Kre09]).

Nicht jede Technik, kann den Zweck zur politischen und sozialen Kontrollen erfüllen.

2.3 TECHNIKEN

2.3.1 *TCP/UDP Portnummern*

Heutzutage ist es nicht außergewöhnlich gleichzeitig im WWW zu surfen, Internet-Radio zu hören und zu chatten. Der Computer ordnet die eingehenden Datenpakete der Anwendung zu und zwar über die TCP/UDP Portnummern. Die Zuordnung nennt man *demultiplexing* (vgl [Ste93, S. 11]). Die Portnummern stehen sowohl für den Absender als auch für den Empfänger im jeweiligen Transport-Protokoll. Das Feld ist jeweils 16 Bit groß und von der *Internet Assigned Numbers Authority (IANA)* in folgende Bereiche eingeteilt (vgl. [RP94, S. 15–38]):

1. 0–1023 bekannte Portnummern (engl. well-known)
2. 1024–49151 registrierte Portnummern
3. 49152–65535 dynamische und private Portnummern

Die Untersuchung der TCP/UDP Portnummern ist die einfachste und weitverbreiteste Technik um Anwendungen und Protokolle zu identifizieren. Die bekannten Ports sind von der IANA mit einer Anwendung verknüpft. Zum Beispiel hat die Anwendung *Telnet* den Wert *23/tcp*. Ein TCP-Paket mit der Ziel-Portnummer 23, kann demnach als *Telnet* klassifiziert werden. Diese Methode ist anfällig für Fehler und erreicht schnell ihre Grenzen.

In einer 2005 durchgeführten Studie zeigte Moore et al. (vgl. [MP05]), dass die Technik annähernd 70 % des Internet-Verkehrs korrekt erkennt. Bei P2P-Software liegt der Wert gemäß einer Studie von Madhukar et al. (vgl. [MW06]) im Jahre 2006 bei 30–70 %. Ein Grund für die niedrige Erkennungsrate ist die Verwendung von dynamischen Portnummern. Anwendungen benutzen oft andere Portnummern als die bekannten, da bei vielen Betriebssystemen zum Öffnen der Portnummern Administratorrechte erforderlich sind (vgl. [RSSD04]). Einige Portnummern sind auch mehrdeutig, wie etwa der Port 888. Dieser wird benutzt von *CD Database Protocol (CDDBP)*, Access-Builder, sowie auch von den *Redundant Array of Independent Disks (RAID)*-Controller Protokolle 3DM und 3DM2 (vgl. [RSSD04]).

Erschwerend kommt hinzu, dass Anwendungen teilweise gezwungen sind den für WWW reservierten Port 80 zu nutzen. Schuld daran ist die Firewall, die in den meisten SOHO-Routern integriert ist und standardmäßig nur Web-Zugriff erlaubt. Hierdurch sind Programme wie Skype, BitTorrent oder Sofortnachrichten-Dienste (IM) gezwungen die Kommunikation durch Port 80 zu tunneln, da ansonsten die Verbindung fehlschlägt. Dies zeigt, dass bei diesen Programmen eine Relation von Portnummer zu Anwendung fast unmöglich ist. Ähnlich verhält es sich mit Angriffen aus dem Internet: Viele Hintertüren, Würmer oder trojanische Pferde versuchen sich auf dem kompromittierten System so unauffällig wie möglich zu verhalten. Dabei tarnen sie sich als fremde Dienste oder nutzen dynamische Portnummern, um unerkannt zu bleiben. Die aufgeführten Beispiele und Studien zeigen, dass eine Identifizierung mittels TCP/UDP-Portnummern keine zuverlässige und robuste Technik ist.

2.3.2 *Deep Packet Inspection*

Die *Deep Packet Inspection (DPI)* ist im Vergleich zur Analyse der TCP/UDP Portnummern eine zuverlässigere Technik. Hierbei wird im ganzen Paket nach spezifischen Mustern gesucht, die mit einer Anwendung verknüpft sind. Die Muster werden durch eine Notation beschrieben, die sich *regulärer Ausdruck* nennt (vgl. [Frio8, S. 1]). Eine gute Analogie in Zusammenhang mit regulären Ausdrücken ist das Textmuster `*.txt`, dass irgendeinen Dateinamen beschreibt, der die Endung `.txt` besitzt. Reguläre Ausdrücke sind aber weitaus komplexer und haben den Umfang einer kleinen Programmiersprache. Die musterbasierte Suche wird oft als *pattern matching* bezeichnet.

Ein Nachteil ist, dass es einer menschlichen Analyse bedarf, um eindeutige Muster zu finden und diese auf dem aktuellen Stand zu halten (vgl. [YSZo8]). Bei älteren Protokollen kann man auf bestehende Listen[2] zurückgreifen; bei neueren Protokollen ist es notwendig, dass Menschen die Dokumentation studieren, um eindeutige Muster zu finden. Dies steht unter der Prämisse, dass die Dokumentation zugänglich ist, was bei proprietären Protokollen nicht der Fall ist. Unter diesem Umstand kann lediglich die Inspektion der rohen Daten Aufschluss über wiederkehrende Muster geben, wobei die Daten lesbar vorliegen müssen.

Ist der Datenverkehr hingegen verschlüsselt oder komprimiert, ist die Technik nicht in der Lage ein eindeutiges Muster zu finden. Dies ist beispielsweise bei Verwendung von *IP security (IPsec)* der Fall. Es handelt sich hierbei um ein Verschlüsselungsprotokoll welches die Integrität, den Datenschutz und die Authentizität sicherstellen soll (vgl. [Tano3, S. 833]). Da die Technik in den

2 http://l7-filter.sourceforge.net/protocols

Inhalt – der *Payload* genannt wird – der Pakete hinein schaut und nach Schlagwörtern sucht, halten viele Kritiker die DPI mit Blick auf den Datenschutz für bedenklich. Ein denkbares Szenario wäre die Erkennung von privaten E-Mails am Arbeitsplatz. Findet der Algorithmus E-Mails mit den Worten *Kuss*, *Schatz* oder *Süß* wird eine Kopie direkt an den Arbeitgeber geleitet. Weiterhin ist die Filterung und Zensur von Inhalten die im Unterabschnitt 2.2.2.2 dargestellt wurde, nur mit dieser Technik möglich. Aus den dargestellten Gründen ist die DPI für den Einsatz bei QoSiLAN keine Option.

2.3.3 *Maschinelles Lernen*

Maschinelles Lernen versucht durch Analyse von Beispieldaten automatisch Gesetzmäßigkeiten zu erkennen und so den entscheidenden Nachteil der DPI, einer menschlichen Datenanalyse, wettzumachen. Maschinelles Lernen kann prädiktiv sein, dass heißt Aussagen über künftige Vorhersagen treffen oder deskriptiv sein, das heißt aufgrund von Beispieldaten neues Wissen erzeugen – oder beides (vgl. [Alp04, S. 3]). Die Technik ist ein sicheres Werkzeug, um Protokolle oder Anwendungen aus IP-Strömen zu identifizieren (vgl. [KCF+08]). Es wird unterschieden in:

ÜBERWACHTES LERNEN: Bei dem überwachten Lernen (engl. supervised learning) muss ein Lehrer die erste Klassifikation übernehmen, damit der Algorithmus aufgrund dieser Daten Muster erkennen kann. Das bedeutet, dass der Algorithmus mit gekennzeichneten Beispieldaten trainiert wird, aus denen er dann Gesetzmäßigkeiten ableiten kann. Bei der Protokoll-Identifizierung entscheidet ein Lehrer vorher in welchem Datenstrom etwa HTTP, FTP oder E-Mail ist und kennzeichnet sie entsprechend.

UNÜBERWACHTES LERNEN: Bei dem unüberwachten Lernen (engl. unsupervised learning) übernimmt der Algorithmus die Klassifizierung. Er erhält unsortierte Beispieldaten und versucht hierin Muster zu finden, um auf ihrer Basis die Daten zu klassifizieren. Bei der Protokoll-Identifizierung bedeutet dass, der Algorithmus muss selbst die Unterschiede erkennen zwischen HTTP, FTP oder E-Mail Daten.

HALB-ÜBERWACHTES LERNEN: Das halb-überwachte Lernen ist eine Mischung aus überwachtem und unüberwachtem Lernen. Es wird eine sehr kleine Menge von gekennzeichneten Beispieldaten benötigt.

Laut Yu Wang et al., brauchen viele dieser Algorithmen abgeschlossene Verbindungen und sind daher nicht geeignet für den

Echtzeitbetrieb: "*Most of these classifiers work offline, since full-flow statistics are not available until a flow is finished, for online deployment it is usually too late to take actions*." ([YSZ08]). Zusätzlich ist die Mathematik hinter den Algorithmen meist sehr komplex und rechenintensiv, was zur Folge hat, dass der Einsatz auf SOHO-Geräten ausgeschlossen ist. Da die Echtzeit-Erkennung einer der Schlüsselanforderungen für die Priorisierung von Strömen bei QoS ist, ist die Technik nicht passend für den Einsatz bei QoSiLAN.

In diesem Kapitel wurde der Grundstein für eine weitere Analyse gelegt und die verschiedenen Möglichkeiten aufgezeigt, um Netzanwendungen zu identifizieren. Weiterhin wurde belegt, warum die Techniken nicht für den Einsatz in der zweite Phase von QoSiLAN geeignet ist. Im nächsten Abschnitt wird eine neue Herangehensweise mit einer konkreten Implementierung gezeigt.

3 STATISTICAL PROTOCOL IDENTIFICATION

In diesem Kapitel wird der SPID-Algorithmus detailliert dargestellt und somit die Theorie anhand einer konkreten Implementierung in die Praxis überführt. Außerdem wird auf Erweiterungen und Optimierungen eingegangen, die entwickelt wurden, um eine möglichst breite Auswahl an Streaming-Protokollen schnell und sicher zu erkennen.

3.1 ALGORITHMUS DETAILS

Erik Hjelmvik und Wolfgang John entwickelten 2009 den SPID-Algorithmus, dessen Ziel es ist, Netzanwendungen mit Hilfe von Statistik zu identifizieren (vgl. [HJ09]). Die Technik versteht sich als Hybrid-Form und nimmt Elemente aus der DPI und behandelt diese wie statistische Werte. Die Abbildung 3 zeigt die drei Phasen, in denen der SPID-Algorithmus arbeitet.

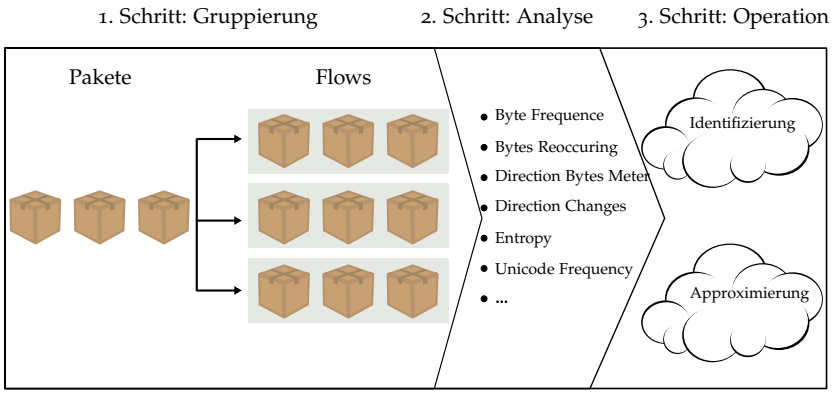

Abbildung 3: Die drei Phasen des SPID-Algorithmus. Quelle: eigene Darstellung.

In der ersten Phase werden einzelne Datenpakete zu zusammenhängende IP-Ströme (engl. flows) gruppiert. Nachdem ein *flow* eine festgelegte Anzahl an Paketen enthält, beginnt die zweite Phase. In dieser Phase entnimmt der Algorithmus die statistischen Proben und wertet diese aus. Die letzte Phase verzweigt sich in die Identifikations- und Lernphase: In der Lernphase werden den Protokollen die statistischen Werte entnommen und in einer Datenbank gespeichert. Diese Werte werden bei der Identifikation mit den gemessen Werten verglichen, woraufhin entschieden wird, ob diese erkannt wurden (vgl. [HJ09]). Jede Phase wird im Nachfolgenden detailliert dargestellt.

3.1.1 *Gruppierung der Datenpakete in Flows*

IP-Ströme können in zwei Richtung verlaufen: Vom Server zum Client und vice versa. Der Algorithmus ordnet jedes Datenpaket einem bidirektionalen *flow* zu, obgleich aus welcher Richtung es kommt. Die Zuordnung eines Datenpakets zu dem entsprechenden *flow* erfolgt über ein 5-Tupel (vgl. [HJ09]) aus:

- Quelladresse
- Zieladresse
- Transport-Protokoll
- Quellport
- Zielport

Die Tatsache, dass ein Paket aus zwei Richtungen kommen kann, hat zur Folge, dass die Technik jedes Paket zweimal prüfen muss. Ein *flow* startet bei dem Transport-Protokol TCP mit dem Dreiwege-Handshake. Dazu schickt entweder der Client oder der Server der Gegenstelle ein TCP-Paket mit einem gesetzten *Synchronize (SYN)*-Flag. Flags sind binäre Zustände die entweder gesetzt oder nicht gesetzt sind. In Abbildung 4 sieht man den Aufbau eines TCP-Pakets. Die Felder in denen der Text vertikal steht, sind die verschiedenen Flags.

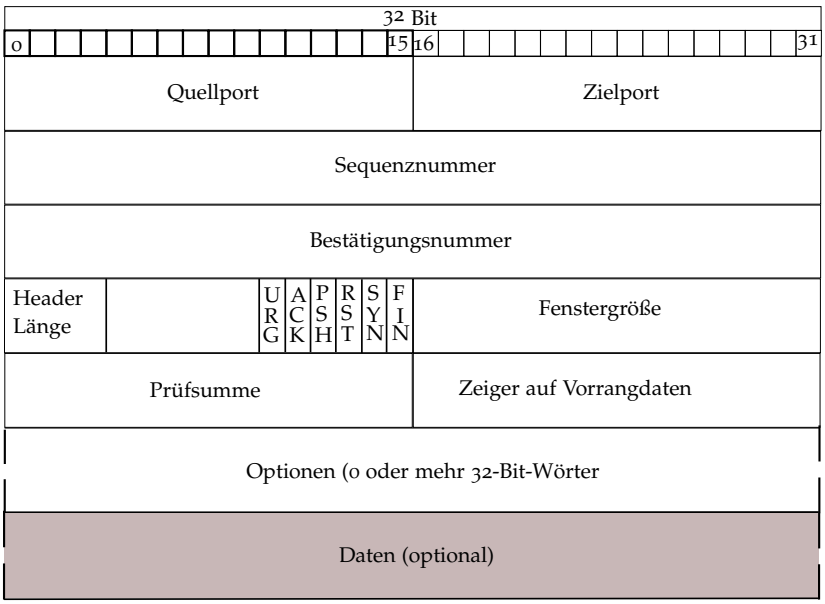

Abbildung 4: Aufbau eines TCP-Pakets. Quelle: [Wik07b] (leicht modifiziert).

Im ersten Paket ist das SYN-Flag und eine beliebige Sequenznummer gesetzt, um der Gegenstelle mitzuteilen, dass der Anfra-

gende eine Verbindung aufbauen möchte. Die Gegenstelle antwortet mit einem Paket mit gesetzten SYN- und *Acknowledgment (ACK)*-Flags, wenn die Verbindung akzeptiert wird. Für denn Fall dass sie abgelehnt wird, schickt die Gegenstelle ein Paket, in dem die *Reset (RST)*- und ACK-Flags gesetzt sind. In beiden Fällen wird die Bestätigungsnummer um eins erhöht. Das ACK-Flag bestätigt immer eine Anfrage. So muss die Bestätigung von dem Teilnehmer, der die Verbindung aufgebaut hat, nochmals mit einem ACK bestätigt werden, wie in Abbildung 5(a) zu sehen ist (vgl. [Ste93, S. 229–233]).

Nachdem die Verbindung erfolgreich mit Hilfe des Dreiwege-Handshake aufgebaut wurde, werden alle nachfolgenden Pakete mit Nutzdaten gesammelt. Das heißt Pakete, die nur das ACK-Flag gesetzt haben und keine Nutzdaten enthalten, nimmt der Algorithmus für den Auf- und Abbau einer Verbindung zur Kenntnis, verwirft sie jedoch anschließend. Wenn eine festgelegte Anzahl an Paketen erreicht ist, nimmt der Algorithmus die statistischen Proben und wertet diese aus. Der Abbau der Verbindung erfolgt ähnlich wie der Aufbau.

Der Client oder Server schickt der Gegenstelle, um eine Verbindung abzubauen, ein Paket mit dem *Finish (FIN)*-Flag. Das FIN-Flag ist das Pendant zum SYN-Flag. Die Gegenstelle beantwortet dies mit einem ACK und einer hochgezählten Sequenznummer. Das Gleiche geschieht nochmal vice versa, wie in Abbildung 5(b) zu sehen ist (vgl. [Ste93, S. 233 ff.]).

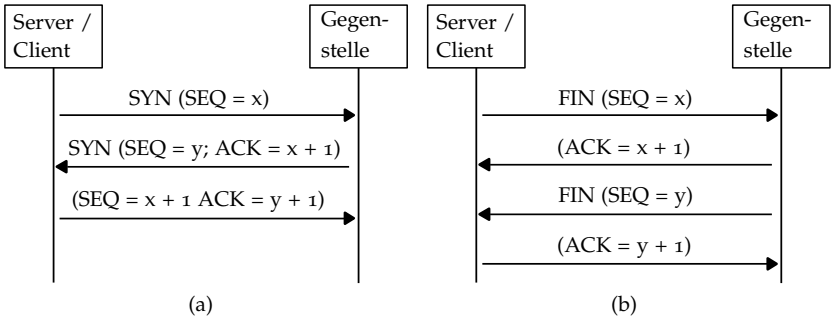

Abbildung 5: (a) Aufbau und (b) Abbau einer TCP-Verbindung. Quelle: eigene Darstellung auf Basis von [Tan03, S. 589 u. 605].

Eine Verbindung ist ebenfalls beendet, wenn ein Teilnehmer eine gewisse Zeit nicht antwortet. Dieser Intervall ist in vielen anderen wissenschaftlichen Arbeiten auf 64 Sekunden gesetzt (vgl. [De 05, S. 3]).

Im Zusammenhang mit dem Dreiwege-Handshake ist das *Kaltstart-Problem* zu nennen. Falls eine Verbindung bereits besteht, verpasst der Algorithmus den Dreiwege-Handshake und nimmt keine Notiz von der Verbindung. Diese Verbindung kann

der Algorithmus somit weder identifizieren noch lernen (vgl. [De 05, S. 3]).

Das Transport-Protokoll UDP ist verbindungslos und braucht daher keinen Verbindungsaufbau oder -abbau. Daher startet ein *flow* mit dem ersten Paket und endet, wenn nach 64 Sekunden kein weiteres Paket ankommt.

Die zweite Phase beginnt, wenn genug Pakete für einen IP-Strom zusammen sind. In diesem arbeiten verschiedene Messverfahren, um die Eigenschaften eines Protokolls statistisch zu erfassen. Hjelmvik benutzte in seinem Prototyp in der Version 0.4 über 30 verschiedene Messverfahren, deren detaillierte Ausführung den Rahmen dieser Arbeit sprengen würde. Eine Darstellung kann jedoch auf der Wiki-Seite des SPID-Projekts nachgelesen werden (vgl. [Hjeo9]). Im Kapitel 3.2.2 wird detailliert auf eine kleine Auswahl von Messverfahren eingegangen, die in der konkreten Implementierung Einzug erhielten.

3.1.2 *Approximierung mit dem empirischen Gesetz der großen Zahlen*

Der SPID-Algorithmus kann aus mitgeschnittenen IP-Strömen *lernen*. Streng genommen, versteht man es nicht als *lernen* wie Menschen und Tiere es tun, sondern als eine Annäherung an einen Wert. Diese Annäherung nennt man in der Mathematik *Approximation*. Die Approximation erfolgt über das empirische *Gesetz der großen Zahlen*. Laut Precht et al. ist das Gesetz wie folgt definiert:

"*Wiederholt man ein zufälliges Experiment genügend oft unter den gleichen Bedingungen, dann kommt die relative Häufigkeit eines bestimmten Ereignisses der theoretischen Wahrscheinlichkeit dieses Ereignisses beliebig nahe.*" ([PKB05, S. 96])

Ein Beispiel eines Zufallsexperiments soll das Gesetz exemplifizieren: Der Münzwurf kann entweder *Kopf* oder *Zahl* ergeben, das heißt die Ergebnismenge entspricht $\Omega = \{K; Z\}$. Laut Bartsch ([Bar04, S. 671]) ist die relative Häufigkeit h_i definiert durch die Formel (3.1).

$$h_i := h_n(x_i) = \frac{k_i}{n} \tag{3.1}$$

hierbei ist

- n Gesamtanzahl
- x Messwert
- k absolute Häufigkeit

Die Wahrscheinlichkeit für Kopf oder Zahl entspricht $P(x_i) = \frac{1}{2}$. Diese Hypothese lässt sich mit dem *Gesetz der großen Zahlen* empirisch prüfen. Dazu wirft man die Münze beliebig oft und errechnet jedes mal die relative Häufigkeit. Die Abbildung 6 zeigt

hundert Durchgänge in Abhängigkeit der relativen Häufigkeit für das Eintreten des Ergebnisses *Zahl*. Die rot gestrichelte Linie zeigt die theoretische Wahrscheinlichkeit von $\frac{1}{2}$ an.

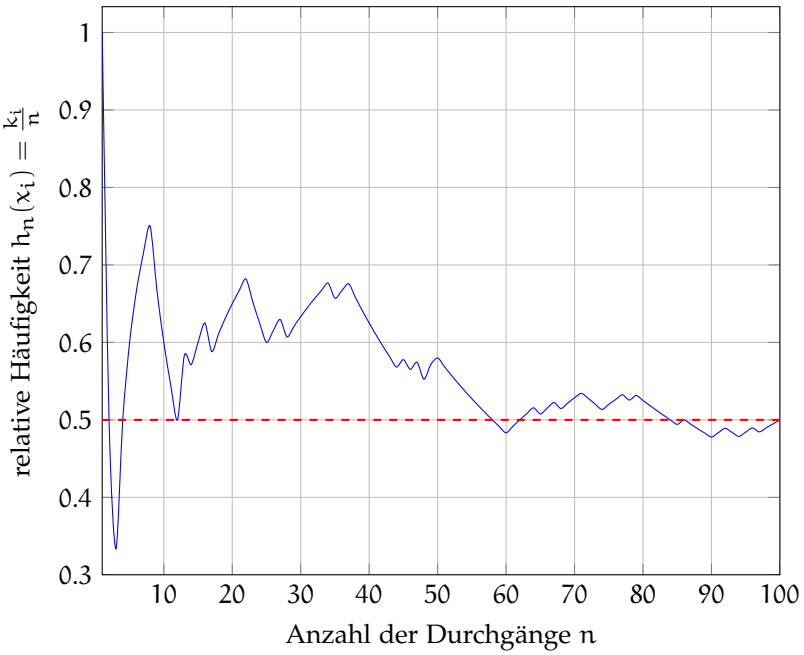

Abbildung 6: Visualisierung des Gesetzes der großen Zahlen. Quelle: eigene Darstellung auf Basis eigener Erhebung.

Dieses Gesetz findet im Lernprozess des SPID-Algorithmus Anwendung. Jede relative Häufigkeit einer Protokoll-Analyse wird mit den vorherigen Durchgängen verrechnet und anschließend in einer Datenbank gespeichert.

3.1.3 *Identifizierung mittels Kullback-Leibler Divergenz*

Die Identifizierung ist das Herzstück des SPID-Algorithmus. Er vergleicht die Wahrscheinlichkeiten der trainierten *flows* (P) mit den gemessenen *flows* (Q) über die *Kullback-Leibler Divergenz (KLD)*, welche in Gleichung (3.2) zu sehen ist (vgl. [KL51]).

$$D(P \parallel Q) = KL(P, Q) = \sum_{x \in X} P(x) \log_2 \frac{P(x)}{Q(x)} \tag{3.2}$$

Die KLD – auch bekannt als *relative Entropie* – ist ein logarithmisches Maß für die Unterschiedlichkeit zweier Wahrscheinlichkeitsverteilungen. Für den Fall dass die Wahrscheinlichkeiten exakt gleich sind, also $P = Q$, gilt $D(P \parallel Q) = 0$, für alle anderen Fälle gilt $D(P \parallel Q) \geqslant 0$. Die Divergenz ist nicht symmetrisch, dass heißt: $KL(P, Q) \neq KL(Q, P)$. Aus informationstechnischer Sicht gibt das Ergebnis der KLD an, wie viel zusätzliche Information nötig ist, um den Wert von (P) zu beschreiben, wenn ein Code mit

Blick auf (Q) optimiert ist. Die Identifizierungsphase vergleicht die Wahrscheinlichkeiten der gemessenen *flows* mit den Protokollen, die in der Datenbank hinterlegt sind. Für jedes dieser Protokolle wird die Kullback-Leibler Divergenz berechnet. Der beste Treffer ist jenes Protokoll, welches die niedrigste Divergenz aufweist. Ist dieser Wert kleiner als ein vorgegebener Grenzwert, ist das Protokoll hierdurch identifiziert. Der Grenzwert hat die Funktion unbekannte Protokolle als eben dieses zu identifizieren.

Eine Alternative zur KLD ist die Kreuzentropie, die in der Gleichung (3.3) zu sehen ist.

$$H(P, Q) = H(P) + D(P \parallel Q) \tag{3.3}$$

Hierbei ist $D(P \parallel Q)$ die KLD und $H(P)$ die Entropie. Die Entropie ist ein Maß für den mittleren Informationsgehalt einer Nachricht. Auf die Entropie wird in Kapitel 3.2.2.5 näher eingegangen. Ein Nachteil dieser Technik ist, dass ein *flow* der einen hohen Entropiewert hat, zugleich einen hohe Kreuzentropie erzeugt und dadurch das Ergebnis verfälscht (vgl. [Hjeo8]). Daher ist diese Gleichung ungeeignet um Protokolle zu identifizieren.

3.2 IMPLEMENTIERUNG

In diesem Kapitel wird eine Software vorgestellt, die den SPID-Algorithmus umsetzt. Die Software wurde so modifiziert und erweitert, dass diese besser skaliert und ein breiteres Spektrum an Protokollen erkennt. Vor allem Streaming-Protokolle standen während der Entwicklung im Mittelpunkt.

Die Software wurde in der Programmiersprache `C++` programmiert, unter Verwendung der libPcap[1]. Mit Hilfe dieser Bibliothek kann man direkt auf die Netzwerkpakete zugreifen, noch bevor sie das Betriebssystem verarbeitet. Zudem steht die Bibliothek für alle gängigen Betriebssysteme zur Verfügung und bietet dadurch eine gute Kompatibilität. In Abbildung 7 ist das Klassendiagramm der Implementierung zu sehen.

Die Klassen SPID und QoSiLAN sind über ein Observer-Entwurfsmuster verbunden. Bei dem Entwurfsmuster gibt es ein beobachtendes Objekt – auch Subjekt genannt – und einen konkreten Beobachter, hier die Klasse `QoSiLAN`. Erkennt die Klasse SPID einen *flow* ruft es die Methode `notifyObservers()` auf, die die Klasse `QosiLAN` über den erkannten *flow* informiert. Möchte ein Objekt die Nachrichten abonnieren, muss es das Interface `IObserver` implementieren.

SPID legt für jedes ankommende Paket eine neue Klasse `IP::V4` an und übergibt dieser das Paket. Die Klasse extrahiert alle benötigten Informationen und legt je nach Bedarf ein neues Objekt

1 http://www.tcpdump.org/

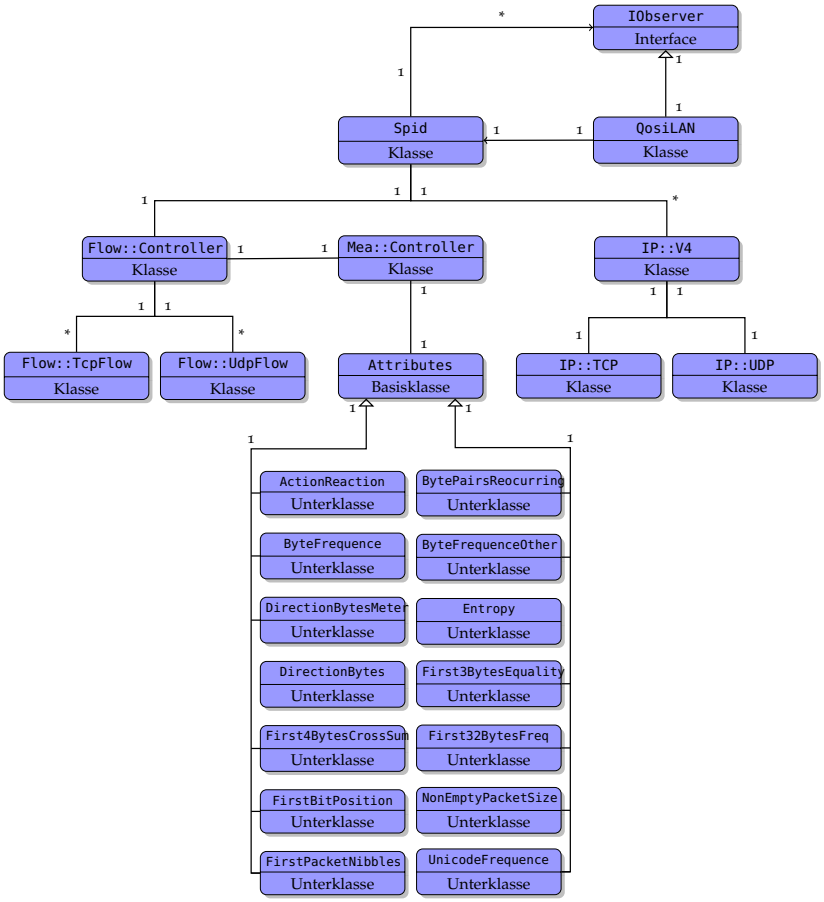

Abbildung 7: Klassendiagramm der SPID-Implementierung. Quelle: eigene Darstellung auf Basis von [Pru09].

der Klassen `IP::TCP` oder `IP::UDP` an. Hier finden neben weiteren Filterungen auch Prüfungen statt, um sicherzustellen, dass es sich um ein valides Paket handelt. Das angelegte `IP::V4`-Objekt wird anschließend dem `Flow::Controller` übergeben. Diese Klasse prüft, ob das Paket anhand der 5-Tupel einem *flow* zugehörig ist. Fällt die Prüfung positiv aus, wird das Paket dem *flow* hinzugefügt; fällt sie negativ aus, wird ein neues `Flow::TcpFlow` oder `Flow::UdpFlow` Objekt erzeugt. Außerdem prüft der `Flow::Controller`, ...

- ... ob ein *flow* länger als 64 Sekunden kein Paket erhalten hat und löscht es entsprechend.
- ... ob ein *flow* genug Pakete für eine Identifizierung enthält.
- ... ob ein *flow* sich im Auf- oder Abbau befindet. Dieser Fall ist nur für TCP relevant.

Sobald ein *flow* genug Pakete für eine Identifizierung enthält, wird dem `Measurement::Controller` der *flow* übergeben. Dieser ruft sequenziell alle Messverfahren auf, die wiederum entnehmen

dem *flow* die statistischen Proben entnehmen. Jedes Messverfahren erbt von der Basisklasse `Measurement::Attributes` Variablen und Methoden. Die Logik des Messverfahren liegt im Konstruktor der jeweiligen Klasse und wird mit Templates – auch Schablonen genannt – implementiert. In der Auflistung 1 sieht man einen exemplarischen Programmcode für ein Messverfahren.

Auflistung 1: Beispiel einer Template-Methode von einem Messverfahren

```
template <class T> BeispielMessverfahren(T* flow) {
    // Größe der Vektoren
    size          = 10;
    // Vektor mit den Häufigkeiten
    count         = new int[size]();
    // Vektor mit den Wahrscheinlichkeiten
    probabilities = new float[size]();

   // Durchlaufe alle Pakete in einem Flow und zähle
   // die Häufigkeiten
   for (int i = 0; i < flow->packets.size(); ++i) {

       if (flow->packets.at(i)->protocol == TCP)
       // Hier ist die Logik für TCP-Pakete
       }
       else {
       // Hier ist die Logik für UDP-Pakete
       }
   }

   // Bezugswert für die Wahrscheinlichkeiten
   total_size = flow->packets.size();

   // Berechnung der Wahrscheinlichkeiten
   for (unsigned short i = 0; i < size; ++i)
       probabilities[i] = count[i] / static_cast<float>(
           total_size);
}
```

Diese Art der Abstraktion ermöglicht es, schnell neue Messverfahren zu programmieren und dadurch den Algorithmus zu erweitern. Nachdem alle Messverfahren ihre Proben aus den Paketen entnommen haben, ruft der `Measurement::Controller` die Methode `IdentifyFlow()` auf, die den IP-Strom letztendlich identifiziert. In Abbildung 8 ist ein Sequenzdiagramm dargestellt, welches die Kommunikation zwischen den Klassen beschreibt. Das Sequenzdiagramm zeigt die Identifizierung eines TCP-Stroms.

Beim Initialisieren der `Spid`-Klasse, legt diese ein Objekt vom Typ `Flow::Controller` an. Diese wiederum legt sich ein Objekt vom Typ `Measurement::Controller` an. Nach der Initialisierung fängt die For-Schleife an, die für jedes Paket durchlaufen wird. Im konkreten Fall handelt es sich um ein TCP-Paket, weswegen die

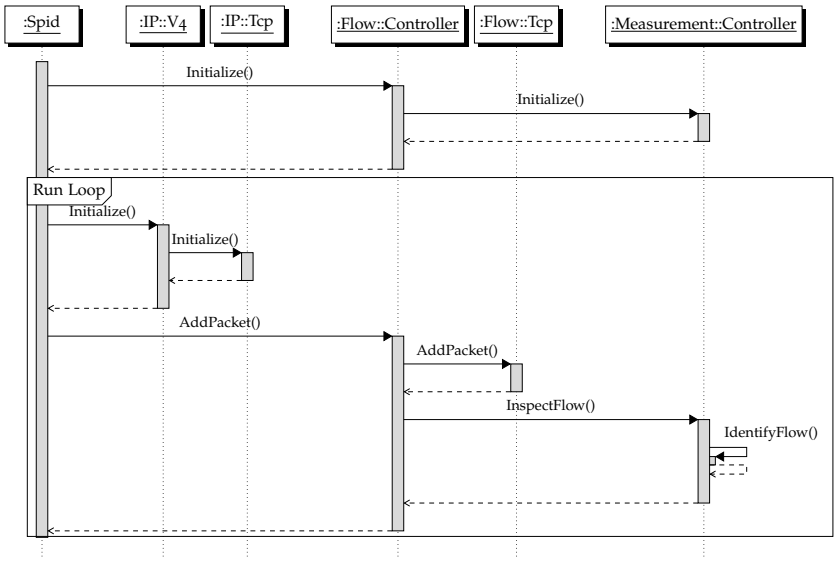

Abbildung 8: Sequenzendiagramm der SPID-Implementierung. Quelle: eigene Darstellung.

`Spid`-Klasse die zwei Klassen `IP::V4` und `IP::Tcp` anlegt. Das Objekt `IP::V4` wird anschließend der Klasse `Flow::Controller` über die Methode `AddPacket()` übergeben. Die fügt das Paket mittels der gleichnamigen Methode dem zugehörigen TCP-*flow* hinzu. Sind alle benötigten Pakete zusammen, ruf `Flow::Controller` die Methode `InspectFlow()` auf, die die Messverfahren ausführt. Nachdem alle Messverfahren ihre Proben entnommen haben, wird der *flow* von der Methode `IdentifyFlow()` identifiziert.

3.2.1 *Neue Leistungsmerkmale*

Bei der Entwicklung der Software sind die folgenden Erweiterungen mit eingeflossen:

UDP UNTERSTÜTZUNG : Der Prototyp in der Version 0.4 von Hjelmvik unterstützt kein UDP. Besonders Streaming-Protokolle nutzen allerdings das Transport-Protokoll UDP um Daten schnell ohne *Overhead* an den Empfänger zu übertragen. Beispiele sind hier: RTP, *Moving Picture Experts Group Transport Stream (MPEG-TS)* und *Trivial File Transfer Protocol (TFTP)*.

ECHTZEIT-ERKENNUNG : Ein weiterer Nachteil ist die fehlende Echtzeit-Erkennung. Dadurch ist die Identifizierung ausschließlich auf mitgeschnittene Datenpakete beschränkt. Bei der neuen Implementierung reicht die Angabe des Netzwerk-Geräts und eventuelle Administratorrechte um

auf dem Gerät zu horchen. Optional lässt sich die Netzwerkkarte in den Promiscuous-Mode[2] versetzen.

OPTIMIERTE DATENBANK : In der Datenbank sind die statistischen Werte der Protokolle gespeichert. Bei Hjelmvik hat diese eine Größe von 9,8 MB für 12 Protokolle. Da bei vielen eingebetteten Systemen und SOHO-Geräten nur wenig Speicherplatz zur Verfügung steht, ist eine geringe Datenmenge von Vorteil. Daher fiel die Entscheidung auf eine optimierte Datenbank, in der sich für jedes Messverfahren ein Vektor mit beliebiger Anzahl an Elementen in Binärform speichern lässt. Diese Form reduziert die Datenmenge auf 389 KB für 17 Protokolle.

AUSGEWÄHLTE MESSVERFAHREN : Aus den über 30 Messverfahren die Hjelmvik entwickelte, wurden diejenigen ausgewählt, die für die Zwecke der Arbeit gute Ergebnisse erzielten. Zusätzlich sind 5 neue Messverfahren hinzugekommen.

3.2.2 *Messverfahren zur Ermittlung der Protokolleigenschaften*

Die nachfolgenden Messverfahren arbeiten auf den ersten 20 Datenpaketen eines *flow*. Diese Anzahl ist mindestens erforderlich, um einen *flow* zu erkennen und erwies sich als zeitlich früh genug, um einen *flow* zu priorisieren. Der Wert wurde durch Experimente mit verschiedenen Werten ermittelt, deren Erkennungsergebnisse daraufhin betrachtet wurden. Dazu mehr in Abschnitt 4.3.2. Alle nachfolgenden Diagramme und Grafiken bauen auf dem Datenbestand auf, die im Kapitel 4.1 detaillierter darstellt wurden.

3.2.2.1 *Byte-Frequenz*

Die Byte-Frequenz – auch Häufigkeitsanalyse genannt – operiert auf dem ersten TCP-Paket jeder Richtung und zählt die relative Häufigkeit eines Bytes in Bezug auf seine Größe (vgl. [HJ09]). Ein Beispiel: Ein 100 Bytes großes Datenpakt einer HTTP GET Anfrage beginnt mit dem Inhalt aus der Auflistung 2.

Auflistung 2: Teil eines HTTP-Headers in Byte- und ASCII-Form

```
47 45 54 20   2F 66 6F 6F   G E T  / f o o
48 54 54 50   2F 31 2E 31   H T T P / 1 . 1
```

Das Messverfahren hat das Byte `0x54` dreimal erfasst und errechnet dadurch eine relative Häufigkeit von $h_{100}(0x54) = 0,3$.

2 Empfangsmodus, in welchem das Gerät den gesamten ankommenden Datenverkehr mitliest, anstatt nur den für ihn Bestimmten.

Zur Speicherung der Ergebnisse der Byte-Frequenz benötigt man einen Vektor mit 256 Werten; da ein Byte aus 8 Bits besteht, ergibt sich hierfür: $2^8 = 256$. Mit der Byte-Frequenz lassen sich markante Häufigkeiten statistisch ermitteln und speichern. Einen deutlichen Unterschied sieht man im Histogramm aus Abbildung 9, bei dem das Klartext-Protokoll HTTP mit einem verschlüsselten oder komprimierten Protokoll wie *Real Time Messaging Protocol (RTMP)* verglichen wird.

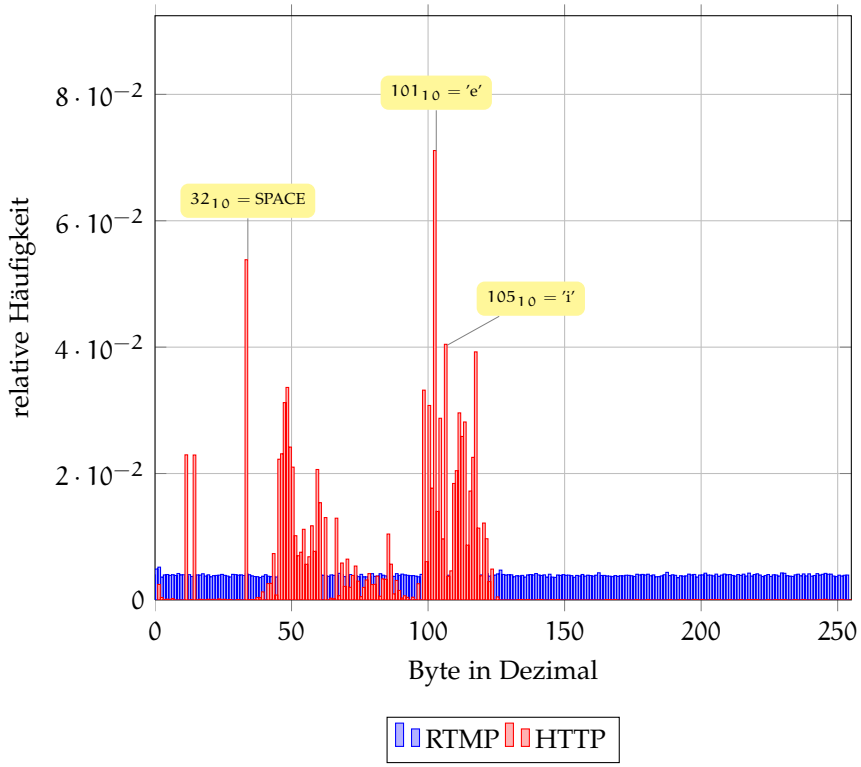

Abbildung 9: Vergleich der Byte-Frequenz. Quelle: eigene Darstellung auf Basis eigener Erhebung.

Das RTMP hat eine hohe Entropie und verteilt sich gleichmäßig über das Histogramm. Das HTTP hingegen, benutzt aufgrund der Zeichencodierung *American Standard Code for Information Interchange (ASCII)* 7 Bits und nimmt daher die Hälfte des Histogramms ein. Zudem kristallisieren sich markante Bytes heraus, die häufiger vorkommen. Die entsprechenden Bytes des Protokolls HTTP aus Abbildung 9 sind in Tabelle 1 aufgeführt.

Ein weiteres Einsatzgebiet der Häufigkeitsanalyse ist das Entziffern von symmetrischen Verschlüsselungen wie dem Substitutionsverfahren. Ein Beispiel für ein Substitutionsverfahren ist die *Cäsar-Chiffre*. Hierbei wird jeder Buchstabe im Alphabet um eine Zahl n verschoben. Die Häufigkeitsanalyse identifiziert die Buchstaben, die besonders häufig vorkommen. Die gewonnenen Daten lassen sich mit Wahrscheinlichkeitsverteilungen für verschiede-

RANG	BYTE	ZEICHEN	ANZAHL	HÄUFIGKEIT
1.	$65_{16} = 101_{10}$	'e'	2667	0.0711143
2.	$20_{16} = 32_{10}$	SPACE	2019	0.0538357
3.	$69_{16} = 105_{10}$	'i'	1517	0.0404501

Tabelle 1: Markante Bytes des HTTP. Quelle: eigene Erhebung.

ne Sprachen vergleichen. Das Verfahren macht es möglich die Verschlüsselung peu à peu zu entziffern. (vgl. [Scho7, S. 45–46]).

3.2.2.2 *Byte-Frequenz der ersten 32 Bytes*

Dieses Messverfahren ist identisch mit dem in Abschnitt 3.2.2.1 vorgestellten, mit dem Unterschied, dass dieses Messverfahren nur die ersten 32 Bytes in einem Paket untersucht. Der Hintergrund ist der, dass viele Netzwerk-Protokolle die UDP nutzen, wenig Klartext-Informationen enthalten und einen kleinen *header* besitzen. Dadurch errechnet sich für die Byte-Frequenz über das ganze Paket eine hohe Divergenz die mit diesem Messverfahren reduziert wird.

3.2.2.3 *Anzahl der Richtungswechsel*

Dieses Verfahren misst die Anzahl der Richtungswechsel in den ersten Datenpaketen. Bei n Paketen sind $(n-1)$ Richtungswechsel möglich. In Abbildung 10 sieht man die durchschnittliche Anzahl der Richtungswechsel bei 20 Paketen der untersuchten Protokolle.

Es zeigt sich, dass interaktive Protokolle wie etwa *Telecommunication Network (Telnet)*, FTP oder SSH häufiger Richtungswechsel haben, als beispielsweise Streaming-Protokolle wie RTP oder *Windows Media Video (WMV)*- und OGG-Streams. Ebenfalls ersichtlich wird, dass *Microsoft Media Server Protocol (MMS)*, RTMP und das Chat-Protokoll *Internet Relay Chat (IRC)* ungefähr die Hälfte an Richtungswechseln haben und dadurch die Kommunikation zwischen Server und Client ausgeglichen ist.

3.2.2.4 *Datenmenge pro Richtung*

Das Verfahren misst die Datenmenge je Richtung und setzt diese in ein prozentuales Verhältnis. In Abbildung 11 sieht man die Verteilung, bei den jeweiligen Protokollen.

Die Protokolle MPEG-TS, RTP und *Simple Mail Transfer Protocol (SMTP)* heben sich von anderen ab, da die Datenmenge die vom Absender zum Empfänger verschickt wird, höher ist als umgekehrt. Das hat folgende Gründe: Das Ziel von SMTP ist es, eine

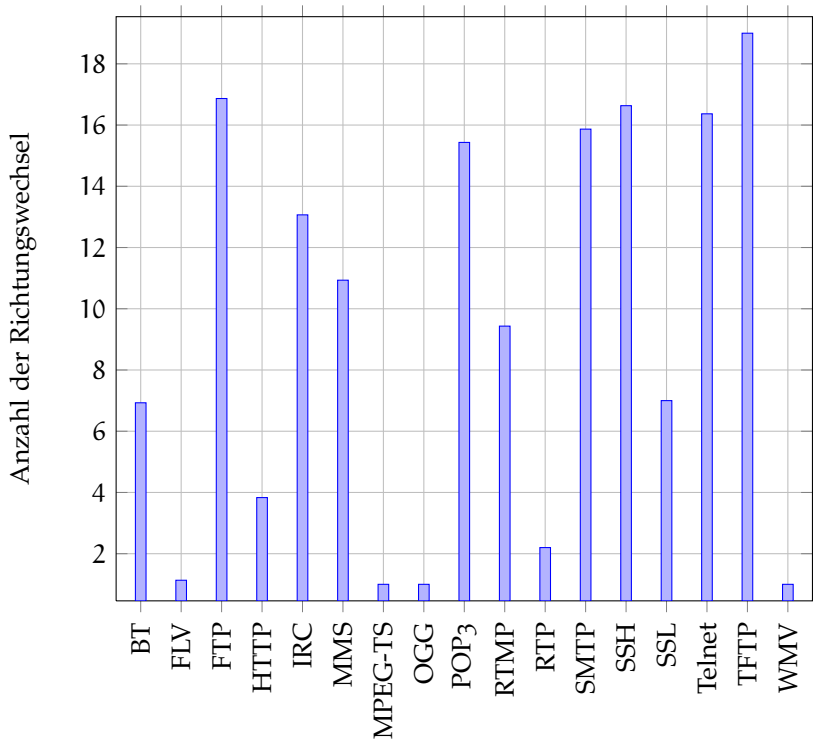

Abbildung 10: Durchschnittliche Anzahl der Richtungswechsel. Quelle: eigene Darstellung auf Basis eigener Erhebung.

E-Mail zu verschicken, daher ist das Datenaufkommen in Richtung des Empfängers größer. Bei MPEG-TS und RTP initiieren im Unterschied zu anderen Protokollen der Server die Verbindung und schickt die Daten an:

- einen Teilnehmer (Unicast)
- n Teilnehmer (Multicast)
- alle Teilnehmer (Broadcast)

Dieses Verhalten spiegelt sich auch in Abbildung 11 wieder, aus der klar ersichtlich hervorgeht, dass die Datenmenge in der Richtung vom Absender zum Empfänger höher ist, als andersherum. Das Messverfahren kann zwischen den folgenden Eigenschaften differenzieren:

- Upload und Download ausbalanciert (z.B. RTP bei *Voice over Internet Protocol (VoIP)*)
- fast ausschließlich Download (z.B. HTTP, *Post Office Protocol Version 3 (POP3)*, ...)
- fast ausschließlich Upload (z.B. SMTP)

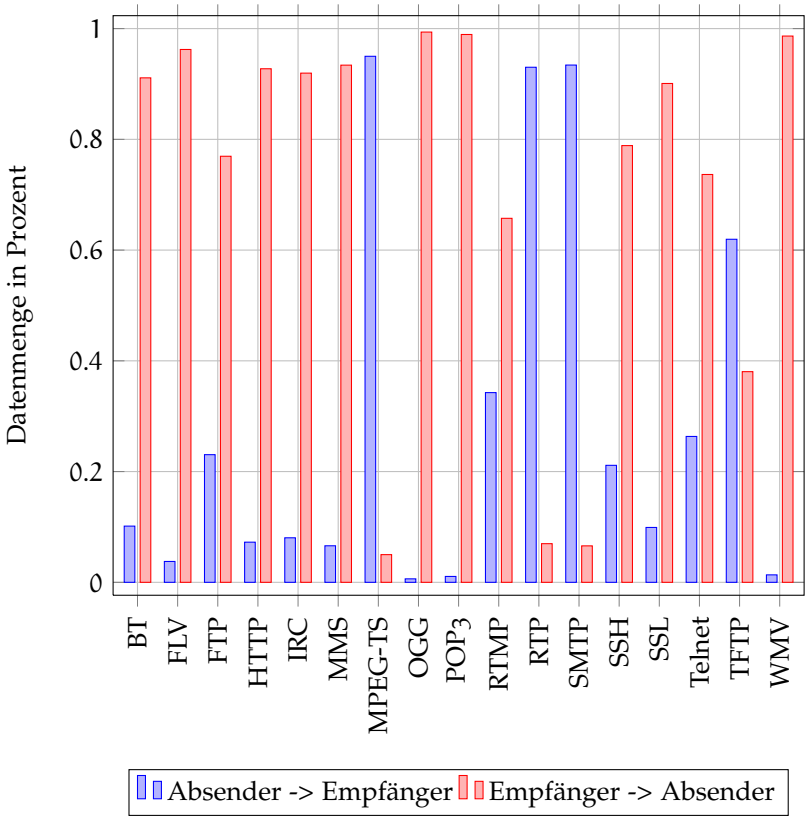

Abbildung 11: Durchschnittliche prozentuale Datenmenge pro Richtung. Quelle: eigene Darstellung auf Basis eigener Erhebung.

3.2.2.5 *Entropie*

Die Entropie ist ein Maß für den mittleren Informationsgehalt einer Nachricht (vgl. [Sha48]). Aus informationstheoretischer Sicht gibt die Entropie an, wieviel Bits mindestens notwendig sind, um ein Codewort zu speichern. Des Weiteren spezifiziert die Entropie, wieviel Zufallsinformation in einer Nachricht enthalten ist. Shannon definiert die Entropie durch die Formel (3.4).

$$H(I) = -\sum_{i=1}^{N} p_i \cdot \log_2 p_i \tag{3.4}$$

hierbei ist

- I Information/Nachricht
- N Alphabet
- p_i Wahrscheinlichkeit eines Zeichens

Umso weniger Redundanz in einer Nachricht enthalten ist, umso höher ist die Entropie. Da die natürliche Sprache durch hohe Redundanz geprägt ist, errechnet sich für Datenpakete in denen diese enthalten ist eine niedrigere Entropie. Bei Datenpaketen mit hoher Entropie lässt sich darauf schließen, dass diese

verschlüsselt oder komprimiert sind. Treten alle vorkommenden Zeichen mit der gleichen Wahrscheinlichkeit auf, ergibt sich der maximale Informationsgehalt H_0. Dieser errechnet sich durch die Formel (3.5).

$$H_0 = \log_2 N \tag{3.5}$$

Bei $N = 256$ ergibt sich eine maximale Entropie von $H_0 = 8$. Da die Häufigkeit eines Bytes mit der Methode der Byte-Frequenz wie in Abschnitt 3.2.2.1 dargestellt bereits berechnet wurde, können diese Ergebnisse in die Berechnung der Entropie einfließen und so Rechenzeit sparen. Die Entropie operiert auf dem ersten TCP-Paket jeder Richtung, da die vollständige Byte-Frequenz ausschließlich bei TCP angewendet wird. In Abbildung 12 ist jeweils die Entropie für jede Richtung ersichtlich.

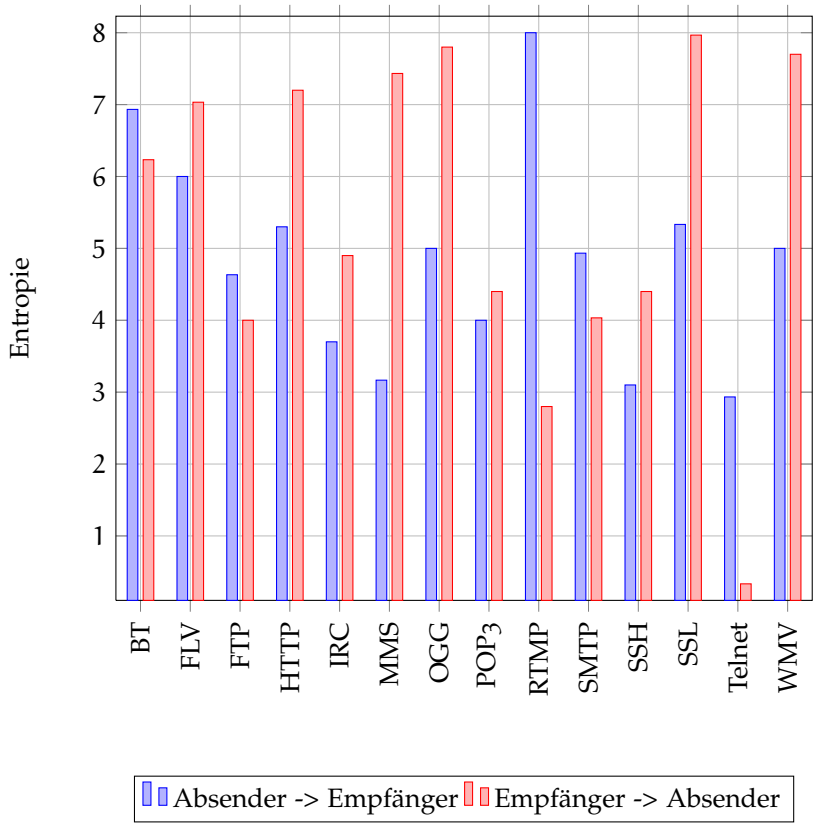

Abbildung 12: Durchschnittliche Entropie beim ersten Paket jeder Richtung. Quelle: eigene Darstellung auf Basis eigener Erhebung.

Auffallend niedrige Entropiewerte erreichen die Protokolle IRC, Telnet, POP3, SMTP und FTP. Viele der Protokolle versenden in den ersten Paketen Steuerkommandos, die in natürlicher Sprache vorliegen und dementsprechend nicht zufällig gewählt sind. Hingegen errechnet sich ein hoher Entropiewert für die Protokolle RTMP, *Secure Sockets Layer (SSL)*, WMV- und OGG-Streams. Die Daten in den Protokollen sind komprimiert oder verschlüsselt.

3.2.2.6 *Hashfunktion über die ersten vier Bytes*

Die ersten Bytes geben in hohem Maß Aufschluss über den Inhalt des Pakets. Dieses Messverfahren nimmt die ersten vier Bytes der ersten zwei Pakete und berechnet dazu einen Hashcode. Ein Hashcode ist das Ergebnis einer Hashfunktion, deren Ziel es ist, einen Code zu errechnen, der die Eingabe möglichst eindeutig beschreibt. Eine einfache Variante der Hashfunktion ist die Quersumme. Sie wird verwendet, um die ersten vier Bytes eines Paketes zu beschreiben. In Auflistung 3 ist ein Beispiel der Berechnung des Protokolls HTTP zu sehen.

Auflistung 3: Beispiel der Quersumme bei den ersten 4 Bytes von HTTP

```
47 45 54 20   G E T    = 31
48 54 54 50   H T T P  = 35
```

Die Quersumme ist eine Hashfunktion die nicht kollisionssicher ist, das heißt, es gibt mehrere Bytefolgen, die beispielsweise die Quersumme 31 erzeugen. Dafür ist die Quersumme äußerst schnell zu berechnen und platzsparend zu speichern. Die höchste Quersumme bei 256 (1 Byte) ist die Zahl 199. Die Quersumme von $199 = 1 + 9 + 9 = 19$. Dieser Wert mit vier multipliziert ergibt einen Vektor mit 76 Elementen, in dem die Häufigkeiten gespeichert sind.

3.2.2.7 *Aktion und Reaktion*

Das Messverfahren untersucht aufeinanderfolgende Pakete, die nicht in die gleiche Richtung gesendet worden sind; das heißt all jene Punkte, an denen ein Richtungswechsel stattfindet. Bei diesen Paketen berechnet das Verfahren die Quersumme der ersten drei Bytes und speichert das Ergebnis. Hintergrund ist der, dass einige Protokolle auf gewisse Aktionen mit wiederkehrenden Reaktionen antworten. Beispielsweise sendet ein IRC-Server Ping-Pakete in regelmäßigen Abständen an den Client, um zu prüfen, ob dieser noch erreichbar ist. Falls er das ist, antwortet er mit einem Pong-Paket. In diesem Fall misst das Verfahren die ersten drei Bytes der Anfrage `PIN` und der Antwort `PON`. Bei HTTP sendet der Client eine Get-Anfrage an den Server, der wiederum antwortet mit einem HTTP-Paket. Das Messverfahren misst hierbei `GET` und `HTT`. Es arbeitet besonders gut bei kommandobasierten Protokollen wie FTP, IRC und POP3.

3.2.2.8 *Periodizität von Byte-Paaren*

Die Periodizität von Byte-Paaren analysiert die Periode, in welcher wiederkehrende Bytes auftreten. Das Messverfahren analysiert die ersten 32 Bytes der ersten vier Pakete und identifiziert beispielsweise das *SS* im SSH-Banner, *22* bei FTP-Servern oder *gg*

bei Ogg-Streams. Gespeichert wird sowohl das Byte, als auch die Anzahl der Bytes die dazwischen liegen. Das Verfahren kann somit auch den Buchstaben *t* im Wort *Torrent* erkennen.

3.2.2.9 *Similarität der ersten drei Bytes*

Einige Protokolle die UDP als Transport-Protokoll nutzen, wie etwa RTP, TFTP oder MPEG-TS besitzen einen fixen *Header*, der in jedem Datenpaket enthalten ist. Das Verfahren prüft in allen untersuchten Paketen die ersten drei Bytes und wertet aus, inwieweit diese identisch sind. Es wird nicht analysiert welche Bytes identisch sind, sondern lediglich ob diese gleich sind. Ein Beispiel: Von 20 untersuchten Paketen des Protokolls RTP, fangen 18 und somit 90 % mit der Byte-Folge `80 60 2E` an. Das Verfahren findet nur bei dem Transport-Protokoll UDP Einsatz.

3.2.2.10 *Unicode Frequenz*

Unicode ist ein internationaler Standard zur Kodierung von Zeichen und hat zum Ziel, jedes in Gebrauch befindliche Zeichen zu kodieren. Die Unicode-Frequenz sucht in den ersten fünf Paketen nach Unicode-Zeichen und misst die Häufigkeit im Vergleich zur Paketgröße. Einige Protokolle wie etwa MMS oder WMV-Streams enthalten nämlich Unicode-Zeichen (vgl. [Mic10, S. 26]). In Abbildung 13 sieht man einen Ausschnitt aus *Wireshark*[3], in welchem ein Teil des *headers* von MMS dargestellt ist.

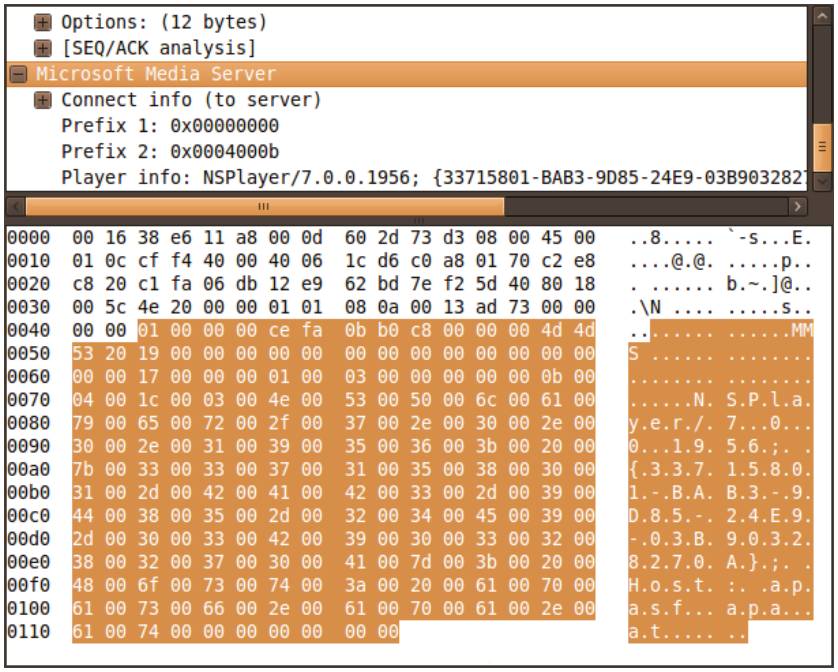

Abbildung 13: Ausschnitt aus Wireshark. Quelle: eigene Darstellung.

3 Ein Programm zur Analyse von Netzwerkverkehr http://www.wireshark.org

Auffallend in der Abbildung 13 ist, dass das Bytes `0x00` häufig vorkommt. Dies ist charakteristisch für Unicode, da die basislateinischen Zeichen mit zwei vorangestellten Nullen beginnen: `0000–007F`. Das nachfolgende Byte entspricht den normalen ASCII-Zeichen. Das Messverfahren bemisst und analysiert die basislateinischen Unicode-Zeichen der Pakete und speichert das Ergebnis.

3.2.2.11 *Bit-Frequenz und Offset*

Besonders UDP-Anwendungen haben einen kleinen *header*, in welchem die Felder und Flags nur ein paar Bits einnehmen. Dieses Faktum macht es der Byte-Frequenz schwer diese zu bemessen. Das Messverfahren arbeitet auf den ersten vier Bytes aller zu untersuchenden Pakete. Es misst die Häufigkeit der einzelnen Bits in Verbindung mit dessen Position, die *Offset* genannt wird. In Tabelle 2 ist eine exemplarische Darstellung von aufeinanderfolgenden Bytes (y) und dessen Bit-Form (x).

Byte y	1								...								n							
Bit x	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8

Tabelle 2: Exemplarische Darstellung von aufeinanderfolgenden Bytes. Quelle: eigene Erhebung.

Jedes Bit x kann entweder 0 oder 1 sein. Um die statistische Verteilung von 4 Bytes (32 Bits) in einem Vektor zu speichern, sind somit 64 Werte notwendig. Um die Häufigkeiten der einzelnen Bits zu bewerten, lassen sich die einzelnen Stellen des Vektors durch die Formel (3.6) indexieren.

$$I_i = \sum_{i=1}^{32} i \cdot 2 + x_i \tag{3.6}$$

Ein Beispiel: Das Streaming-Protokoll RTP enthält in den ersten zwei Bits die Versionsnummer, die in den meisten Fällen die 2 trägt (vgl. [SCFJ03, S. 12]). Im Binärsystem hat die 2 folgenden Stellenwert: $2_{10} = 10$. In der Programmiersprache `C++` fängt ein Vektor (Array) bei 0 an, daher sind beim ersten Bit $y_i = x_i = 0$. Für das erste und zweite Bit ergibt sich dadurch:

$$I_0 = 0 \cdot 2 + 1 \qquad I_0 = 1$$

$$I_1 = 1 \cdot 2 + 0 \qquad I_1 = 2$$

Der Vektor, der die Wahrscheinlichkeiten speichert, enthält einen hohen Wert bei den Felder eins und zwei.

3.2.2.12 *Payload-Größe des ersten Pakets*

Das Messverfahren untersucht nicht den Inhalt eines Pakets, sondern nur dessen Größe. Die ersten Pakete enthalten Informationen über die Initialisierung des Protokolls und haben in vielen Fällen eine minimale und maximale Größe. De Montigny-Leboeuf hat dieses Kriterium zur Charakterisierung von Protokollen untersucht (vgl. [De 05, S. 28–36]).

Das Verfahren misst die Größe des ersten Pakets und stellt diese der maximalen Paketgröße bei TCP und UDP gegenüber. Die maximale Paketgröße ist bei TCP 1460 Bytes und bei UDP 1476 Bytes, wenn von Jumbo-Paketen abgesehen wird. Die Größen errechnen sich durch Abzug der Größe des *headers* von IP (20 Bytes) und entweder TCP (20 Bytes) oder UDP (8 Bytes) von der *Maximal Transmission Unit (MTU)* von *Ethernet* (1500 Bytes). Abbildung 14 zeigt die durchschnittlichen Paketgrößen einiger Protokolle.

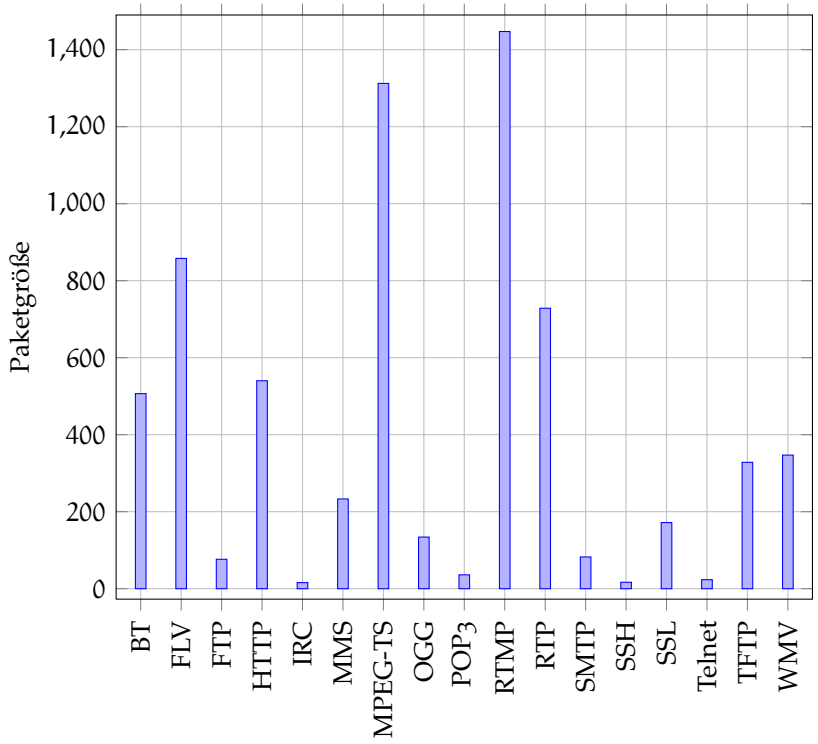

Abbildung 14: Durchschnittliche Paketgröße des ersten Pakets. Quelle: eigene Darstellung auf Basis eigener Erhebung.

Die Protokolle RTMP und MPEG-TS stechen klar mit ihrer durchschnittlichen Paketgröße hervor. Interaktive Protokolle wie IRC, SSH und Telnet haben dagegen nur eine recht kleine Paketgröße.

3.2.2.13 *Messverfahren die nicht integriert wurden*

Bei der Implementierung diverser Messverfahren haben die Ergebnisse gezeigt, dass einige unpräzise sind und als Folge eher

zu einer Verschlechterung führen. Hierunter fallen die folgenden Verfahren:

ZWISCHENZEITEN DER PAKETE: Bei den durchschnittlichen Zwischenzeiten der Pakete wird die Zeit gemessen, die die Pakete zur Übertragung benötigten. Da dieser Wert stark von der Auslastung des Netzes abhängt, hat dieses Verfahren zu einer Verschlechterung der Identifizierung geführt.

PORTNUMMERN: Der Quell- und Zielport kann als statistischer Wert fungieren. Es ist zwar nicht zwingend erforderlich, dass Protokolle die für sie vorgesehene Portnummern benutzen, aber der Großteil der Protokolle tut dies. Sobald der Client oder der Server dynamische Portnummern verwendet, weichen die Werte stark voneinander ab, so dass es zu einer Verschlechterung der Identifizierung gekommen ist.

PAKET-GRÖSSE: Dieses Verfahren sichtet die Größe aller untersuchten Pakete und berechnet ihre Häufigkeit. Da die Paketgröße stark von dessen Inhalt abhängt, ist es zu keiner Verbesserung der Identifizierung gekommen.

DAUER DES FLOWS: Die Dauer wird berechnet indem man von der Ankunftszeit des ersten Pakets die Ankunftszeit des letzten Paketes subtrahiert. Gleichermaßen wie die Zwischenzeiten von Paketen, ist die Dauer eines *flows* abhängig von der Netzinfrastruktur und deren Auslastung. Daher ist es zu übermäßigen Schwankungen gekommen, die sich in einer schlechteren Identifizierung niederschlugen.

3.2.3 *Darstellung der Fingerprint-Datenbank*

Damit der SPID-Algorithmus die Protokolle identifizieren kann, muss er die von den Messverfahren ermittelten Annährungswerte speichern. Die Speicherung erfolgt in einem kompakten Binärformat, wie in Kapitel 3.2.1 besprochen. Die Anzahl der Werte in einem Vektor richtet sich nach dem entsprechenden Messverfahren. Beispielsweise benötigt die Entropie nur einen Wert und nimmt daher auch nur einen Wert in der Datenbank ein; die Byte-Frequenz hingegen benötigt 256, nimmt daher 256 Werte ein.

Grundlegend gibt es drei Vektoren die zur Identifizierung notwendig sind, und in denen folgende Werte gespeichert sind: die Gesamtanzahl, die relative und absolute Häufigkeit. Die Werte ergeben sich aus der Formel (3.1). Durch Umstellen der Formel

lässt sich der Vektor, in welchem die Gesamtanzahl n steht, berechnen.

$$h_i = \frac{k_i}{n} \rightarrow n = \frac{k_i}{h_i}$$

Die Umstellung führt dazu, dass für alle Messverfahren der Vektor für die Gesamtanzahl berechnet und nicht gespeichert wird. In Tabelle 3 sieht man ein Beispiel aus der Fingerprint-Datenbank, die Werte für HTTP aus der Byte-Frequenz enthält.

Index	...	$99_{10} =' c'$	$100_{10} =' d'$	$101_{10} =' e'$	...
Häufigkeiten	...	506	348	1176	...
Wahrscheinlichkeiten	...	0.0312	0.0222	0.0726	...

Tabelle 3: Beispiel der Fingerprint-Datenbank für HTTP. Quelle: eigene Erhebung.

Ein Mangel, der sich aus der Speicherung in Binärform ergibt, ist, dass die Reihenfolge der Bytes plattformspezifisch ist. Jede Rechnerarchitektur kann Bytes verschiedenartig einlesen, dennoch haben sich zwei Formen durchgesetzt: Bei *Little Endian* liest der Computer das niederwertigste Byte zuerst ein und bei *Big Endian* das höchstwertige Byte. Vergleichbar ist das Problem der *Endianness* mit der Richtung der Schreibsprachen, wonach beispielsweise die lateinische Schrift von links nach rechts geschrieben wird und arabische und hebräische Schriften von rechts nach links. Das hat zur Folge, dass die Datenbank umgewandelt werden muss, bevor sie auf einer anderen Computerarchitektur eingesetzt wird.

Zusammengefasst wurde in diesem Abschnitt eine Implementierung des SPID-Algorithmus vorgestellt und die vorgenommenen Modifikationen dargestellt. Des Weiteren wurden die Messverfahren dargelegt, die die Eigenschaften der Protokolle in statistische Annäherungswerte umwandelt. Es folgt die Evaluation der Implementierung, in welcher die Ergebnisse dargestellt werden.

4 EVALUATION DES ALGORITHMUS

Im folgenden Abschnitt wird gezeigt, wie und unter welchen Umständen die Testdaten der Protokolle mitgeschnitten und welche Anwendungen eingesetzt wurden. Des Weiteren werden die Methoden offen gelegt, unter denen die Implementierung evaluiert wurde. Abschließend werden die Ergebnisse präsentiert, die die Implementierung erzielt hat und im Hinblick auf die Zielsetzung der Arbeit bewertet.

4.1 DATENBESTAND

Ein essentieller Teil der Arbeit war es, von jedem Protokoll unterschiedliche Daten zu sammeln, um ein möglichst neutrales Ergebnis zu erzielen. Konkret bedeutet das, dass auch unterschiedliche Software für Server und Client getestet wurde, um eine Mannigfaltigkeit zu erreichen. Umso breiter der Datenbestand, umso aussagekräftiger ist letztendlich das Ergebnis.

4.1.1 *Beschaffung aus frei verfügbaren Quellen*

Aufgrund der Tatsache, dass die zur Verfügung stehenden Mittel begrenzt waren und um den Aufwand so gering wie möglich zuhalten aber dennoch ein breites Feld an Protokollen abzudecken, wurde auf frei verfügbare Datenquellen[1][2][3] zurück gegriffen. Aus diesen ließen sich die benötigen Protokollen mit einem Perl-Script extrahieren. Dadurch war es möglich Mitschnitte von Anwendungen zu erhalten, die ansonsten nur schwerlich zu testen sind.

4.1.2 *Erstellung von Mitschnitten*

Darüber hinaus war es notwendig, eigene Mitschnitte unter Laborbedingungen zu erstellen und zwar mit den Programmen `Tcpdump` und `Wireshark`. Viele Protokolle erforderten wiederholte Abläufe, die mittels der Programmiersprache Perl automatisiert wurden. Insgesamt kamen 3135 reine *flows* für 17 Protokolle zusammen, die mehr als 1.5 GB Datenvolumen ergaben. Im Folgenden wird detailliert dargelegt, welche Anwendungen wie getestet wurden.

1 http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html
2 http://www.openpacket.org
3 http://www.pcapr.net

4.1.2.1 *BitTorrent*

BitTorrent ist ein P2P-Protokoll, das Bram Cohe zum schnellen Austausch von großen Daten entwickelte (vgl. [McC01]). Viele Clients unterstützen die *Message Stream Encryption (MSE)*, die das Ziel verfolgt, den *header* und den *payload* der Pakete so zufällig wie möglich aussehen zu lassen, um von Identifzierungs-Systemen unerkannt zu bleiben (vgl. [Vuz10]). Im Labor wurde das Programm *Transmission*[4] – ein plattformunabhängiger und ressourcenschonender BitTorrent-Client – getestet, welcher MSE unterstützt. Das ergab mit den freien Quellen zusammen 197 IP-Ströme.

4.1.2.2 *Flash Videos*

Flash-Videos sind im Bereich Streaming bei den *progressiven Downloads* angesiedelt. Hierdurch kann man ein Video oder Audio abspielen, ohne die komplette Datei zu besitzen. Es wird lediglich der *header* und eine kleine Datenmenge der Datei benötigt; der Rest wird vom Abspielprogramm progressiv, das bedeutet nach und nach, geladen.

Online-Videoportale gibt es mittlerweile in hoher Anzahl, aber nicht jede nutzt Flash-Videos. Von den folgenden Portalen ließen sich unterschiedliche Videos abspielen und der resultierende Netzwerkverkehr mitschneiden:

- YouTube (`http://www.youtube.com`)
- Clipfish (`http://www.clipfish.de`)
- Google Videos (`http://video.google.com/`)
- Metacafe (`http://www.metacafe.com/`)
- MyVideo (`http://www.myvideo.de/`)
- Sevenload (`http://en.sevenload.com/`)
- Revver (`http://www.revver.com/`)

4.1.2.3 *FTP*

Bei FTP gibt es zwei Bereiche:

PORTNUMMER 21 (FTP CONTROL): Auf diesem Port tauscht der Server mit dem Client die Steuerkommandos aus.

PORTNUMMER 20 (FTP DATA): Auf diesem Port übertragen Server und Client die Nutzdaten.

4 `http://www.transmissionbt.com/`

Das Augenmerk dieser Arbeit liegt auf FTP-Control, da bei dem FTP-Data die rohen Daten übertragen werden und diese schwer zu erkennen sind.

Um ein möglichst breites Spektrum an unterschiedlichen Server zu erhalten, wurde der Untersuchung eine Liste von FTP-Servern[5] zu Grunde gelegt, die den anonymen Zugriff erlauben. In dieser sind 5593 Server aufgelistet. Da die Liste jedoch veraltet war, wurde ein Skript entwickelt, dass die FTP-Server sequentiell durchprobierte und jene ausgefiltert hat, die nicht funktionieren. Letztlich sind noch 1100 Server übrig geblieben.

Die gefilterte Liste wurde einem weiteren Script übergeben, welches sich der Reihe nach auf den Servern einloggte und per Zufall gültige Kommandos absetzte. Parallel dazu wurde der Datenverkehr mitgeschnitten.

4.1.2.4 *HTTP*

HTTP ist ein Protokoll, dass eher lose spezifiziert und daher schwer zu identifizieren ist. Um ein breites Feld abzudecken, wurden verschiedene Webseiten mit unterschiedlichen Browsern aufgerufen und parallel der Datenverkehr mitgeschnitten. Zum Einsatz kamen hierbei die Browser: Firefox/2.0.0.20, Firefox/3.5.7, Opera/9.52 und Mozilla/4.04 [en]. Das ergab mit den freien Quellen zusammen 127 IP-Ströme.

4.1.2.5 *IRC*

Bei dem IRC-Client *XChat* ist eine Liste mit 76 Servern beigelegt, die manuell durchprobiert und den Datenverkehr parallel mitschnitten wurde.

4.1.2.6 *MMS*

Das von der Firma Microsoft entwickelte MMS-Protokoll zur Übertragung von Multimedia-Streams findet häufig seinen Einsatz bei Internetradio oder Video-Streams. Dem Streaming-Programm *tivion* in der Version 0.0.4 liegt eine umfangreiche Liste mit *Uniform Resource Locators (URLs)*, von Fernsehsendern und Internetradios bei. Diese Liste wurde nach URLs die mit `mms://` begannen gefiltert. Es resultierte eine Auflistung mit 364 URLs zu MMS-Streams. Ähnlich wie bei FTP wurde ein Script geschrieben, welches die URLs an ein Wiedergabeprogramm übergab und nach fünf Sekunden automatisch unterbrach. Dadurch entstanden 252 valide IP-Ströme.

5 http://www.ftp-sites.org/

4.1.2.7 *MPEG-TS und RTP*

Um Datenverkehr für MPEG-TS und RTP zu erzeugen, wurde der Multimedia-Player *VLC*[6] benutzt. Dort wurden verschiedene Video-Dateien ausgewählt und über das Transport-Protokoll UDP gestreamt. Mit einem weiteren Computer wurde der Datenverkehr mitgeschnitten. Dieser Prozess ließ sich schwer automatisieren, weshalb er ausschließlich manuell durchgeführt wurde. Darüber hinaus ließen sich hinsichtlich RTP noch ein paar zusätzliche *flows* mit Hilfe des Internet-Telefonie-Programms *Egika*[7] erzeugen, welches die Audio- und Video-Daten über RTP überträgt. Insgesamt basiert die Untersuchung auf rund 40 MPEG-TS *flows* und 69 RTP *flows*.

4.1.2.8 *OGG-Vorbis Streams*

Bei Streams mit OGG-Vorbis wurde ähnlich vorgegangen wie bei MMS-Protokoll. Die Liste mit OGG-Vorbis-Streams stammt von der Xiph.Org Stiftung[8]. Diese Liste wurde automatisiert abgearbeitet und dabei 124 valide Streams aufgezeichnet.

4.1.2.9 *POP3 und SMTP*

Aufgrund der Tatsache, dass man bei den meisten POP_3- und SMTP-Server Zugangsdaten benötigt, stammen die *flows* für die Protokolle allesamt von frei verfügbaren Quellen.

4.1.2.10 *RTMP*

RTMP ist von der Firma Adobe Systems entwickelt worden und ist ein proprietäres Protokoll um Multimedia-Daten über das Netzwerk zu streamen. Die meisten Server-Implementierungen sind kostenpflichtig und nicht frei verfügbar. Eine Ausname bildet die Implementierung *Red5*[9], die in der Programmiersprache Java geschrieben ist und unter einer Open-Source-Lizenz zur Verfügung steht. Der Server wurde auf einem Test-Rechner installiert und von diesem wurden verschiedene Videos gestreamt. Neben dieser Bezugsquelle für *flows* gibt es einige Internet-Seiten, die ihre Videos über RTMP streamen, und mitgeschnitten werden konnten.

4.1.2.11 *SSH und Telnet*

Mit den Protokollen SSH und Telnet kann man sich auf einen entfernten Computer einloggen und diesen fernsteuern. Ein Großteil

6 http://www.videolan.org/vlc/
7 http://ekiga.org/
8 Gemeinnützige Stiftung, die sich für die Entwicklung und den Einsatz von freien Multimedia-Formaten einsetzt. http://dir.xiph.org/
9 http://wiki.red5.org//

der mitgeschnittenen Flows stammt von Fremdquellen, da auch hier Zugangsdaten zu den Rechnern erforderlich sind. Der geringere Teil stammt von Mitschnitten die unter Labor-Bedingungen erstellt wurden.

4.1.2.12 *SSL*

SSL wird dazu verwendet, um Daten verschlüsselt zu übertragen. Am bekanntesten ist der Einsatz bei HTTP. Dadurch können Webseiten eine verschlüsselte Übertragung anbieten und sind mit `https://` am Anfang der URL gekennzeichnet. SSL kann auch bei anderen Protokollen wie POP3, SMTP, IRC und vielen mehr Verwendung finden. Die *flows* wurden hauptsächlich durch den manuellen Aufruf von Webseiten mit SSL generiert.

4.1.2.13 *TFTP*

TFTP ist ein minimalistisches FTP, das auf dem Transport-Protokoll UDP basiert und mit dem Ziel entwickelt worden ist, Betriebssysteme und Konfigurationen über das Netzwerk zu übertragen. Ein paar Dateien wurden mit der Implementierung TFTPD32[10] getauscht, über die Befehle `PUT` und `GET`. Weiterhin wurden ein Betriebssystem über einen entfernten TFTP-Server gebootet. Hierdurch entstanden 42 verwertbare *flows*.

4.1.2.14 *WMV Streams*

Die Mitschnitte der WMV-Streams kamen ausschließlich aus dem Labor. Dort wurde ein Windows Media Server 2003 eingerichtet und so konfiguriert, dass er Videos per HTTP anbot. Ein zweiter Rechner verband sich mit dem Server und schnitt simultan den Datenverkehr mit. Daraus ergaben sich 35 IP-Ströme.

4.2 EVALUATIONSMETHODEN

Bevor auf die Bewertungskriterien eingegangen wird, ist zu definieren, welche Möglichkeiten der Erkennung es gibt. Falls der Algorithmus das Protokoll korrekt vorhergesagt hat, nennt man das Ergebnis *richtig positiv* mit dem Akronym TP. Ist die Vorhersage falsch, so nennt man das Ergebnis *falsch negativ* mit der Abkürzung FN. Das Ergebnis der Menge an Protokollen, die fälschlicherweise identifiziert worden ist, nennt man *falsch positiv* und kürzt man mit FP ab. Alle anderen Protokolle, die richtig erkannt worden sind, nennt man *richtig negativ* und kürzt man mit dem Akronym TP ab. Zusammengefasst ergeben die in Tabelle 4 abgebildete Konfusionsmatrix ist.

10 http://tftpd32.jounin.net/

	tatsächlich positiv	tatsächlich negativ
positiv vorhergesagt	richtig positiv (TP)	falsch positiv (FP)
negativ vorhergesagt	falsch negativ (FN)	richtig negativ (TN)

Tabelle 4: Konfusionsmatrix für binäre Ergebnisse. Quelle: eigene Darstellung auf Basis von [MRS08, S. 155].

Der Algorithmus ließ sich dadurch evaluieren, indem man ihn anwies den bekannten Datenbestand aus Kapitel 4.1 zu erkennen. Die Ergebnisse die der Algorithmus daraufhin ausgab, wurden aufgezeichnet und verrechnet. In Abbildung 15 ist ein Beispiel zu sehen, für die Evaluation des Algorithmus.

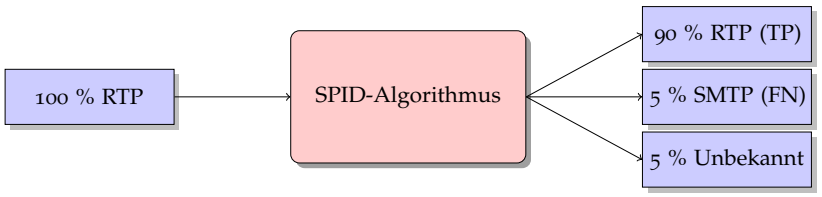

Abbildung 15: Evaluation des Algorithmus. Quelle: eigene Darstellung.

Die Tabelle 4 zeigt, dass es nicht damit getan ist, die Protokolle die der Algorithmus erkennt mit der Trefferquote zu bewerten; es muss ebenso die Genauigkeit betrachtet werden. Eine Trefferquote von 100 % für RTP ist unbrauchbar, wenn der Algorithmus alle Anderen fälschlicherweise ebenfalls als RTP identifiziert. Für eine präzise Evaluation ist daher ein drittes Maß notwendig, dass sowohl die Trefferquote, als auch die Genauigkeit gewichtet. Hierfür dient das harmonische und das arithmische Mittel. Die Formeln (4.1)–(4.4) zur Berechnung dieser Werte stammen aus dem Bereich der Informations-Beschaffung (vgl. [MRS08, S. 155–156]).

$$\text{Trefferquote} = \frac{TP}{TP + FN} \tag{4.1}$$

$$\text{Genauigkeit} = \frac{TP}{TP + FP} \tag{4.2}$$

$$F - \text{Maß} = \frac{2 \cdot \text{Genauigkeit} \cdot \text{Trefferquote}}{\text{Genauigkeit} + \text{Trefferquote}} \tag{4.3}$$

$$\text{Durchschnitt} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{\text{Genauigkeit} + \text{Trefferquote}}{2} \tag{4.4}$$

Das arithmische Mittel – auch Durchschnitt – hat einen entscheidenden Nachteil gegenüber dem harmonischen Mittel: Es behandelt die Trefferquote und die Genauigkeit gleich. Ein Beispiel soll den Punkt konkretisieren. Die Trefferquote liegt bei 100 % und die Genauigkeit bei 1 %. Laut der Formel (4.4) er gibt sich für das arithmische Maß der Wert 50,5 %. Der Wert für das harmonische Mittel hingegen – das auch F-Maß genannt wird – ergibt nach (4.3) 1,9 % und gibt dadurch das Ergebnis realistischer wieder. Daraus folgt, dass das F-Maß besser geeignet ist, um einen Algorithmus zu bewerten der Protokolle identifiziert.

4.3 ERGEBNISSE

Die händische Evaluation von 17 Protokollen und 3135 *flows* ist nicht praktikabel, weshalb ein Script entwickelt wurde, dass diese Arbeit automatisiert. Es arbeitet eine Ordner-Struktur ab und testet dabei alle Mitschnitte die in dem Ordner gespeichert sind. Anschließend wertet das Skript die Daten aus und gibt die Ergebnisse in einer ASCII-Tabelle wieder. In Abbildung 16 sieht man die erzielten Werte, die jeweils mit 30 IP-Strömen approximiert wurden. Die für diese Abbildung zugrunde liegenden Daten befinden sich im Appendix im Kapitel A.

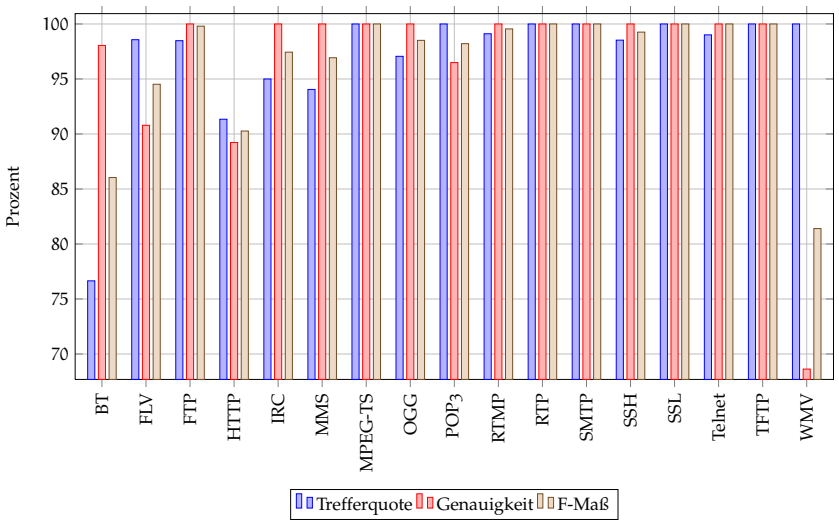

Abbildung 16: Ergebnisse der Protokoll-Identifizierung. Quelle: eigene Darstellung auf Basis eigener Erhebung.

Die Ergebnisse zeigen, dass für jedes untersuchte Protokoll gute Resultate erzielt wurden. Die Identifizierung bei HTTP ist im Vergleich zu den anderen Protokollen schlechter, da HTTP eher vage spezifiziert ist. Schwierigkeiten gibt es bei der Differenzierung zwischen einer Webseite mit Gzip-Kompression oder vielen Bildern und einem Flash-Video. Eine niedrigere Genauigkeit erzielten WMV-Streams, da diese nicht nur durch HTTP getunnelt,

sondern auch oft über MMS gestreamt werden. Aufgrund dieser Tatsache hat die SPID-Implementierung Probleme, diese zu differenzieren. Die Trefferquote von BitTorrent liegt unter dem Durchschnitt, da das Protokoll verschiedene Techniken nutzt, um nicht erkannt zu werden. Hjelmvik et al. entwickelte dafür ein spezielles Messverfahren mit dem Namen *AccumulatedDirectionBytesMeter*, um BitTorrent dennoch zu identifizieren (vgl. [Hje09]). Auf diese Methode wurde jedoch verzichtetet, da es signifikant die Trefferquote und Genauigkeit der anderen Protokolle minderte.

4.3.1 *Ergebnisse in Abhängigkeit der approximierten IP-Ströme*

Die Ergebnisse in Abhängigkeit der gelernten oder präziser der approximierten *flows* sind in Abbildung 17 zu sehen. Erkennbar ist, dass die Identifizierung besser wird, umso mehr *flows* der Algorithmus lernt. Der Graph zeigt, dass ab 20 approximierten *flows* die Erkennung über 80 % liegt. Dieser Wert ist eine kritische Marke für eine stabile Erkennung. Für Protokolle, die wie IRC, Telnet, SMTP, etc. die in Klartext vorliegen, ist eine Identifizierung bereits nach dem 10 *flow* möglich.

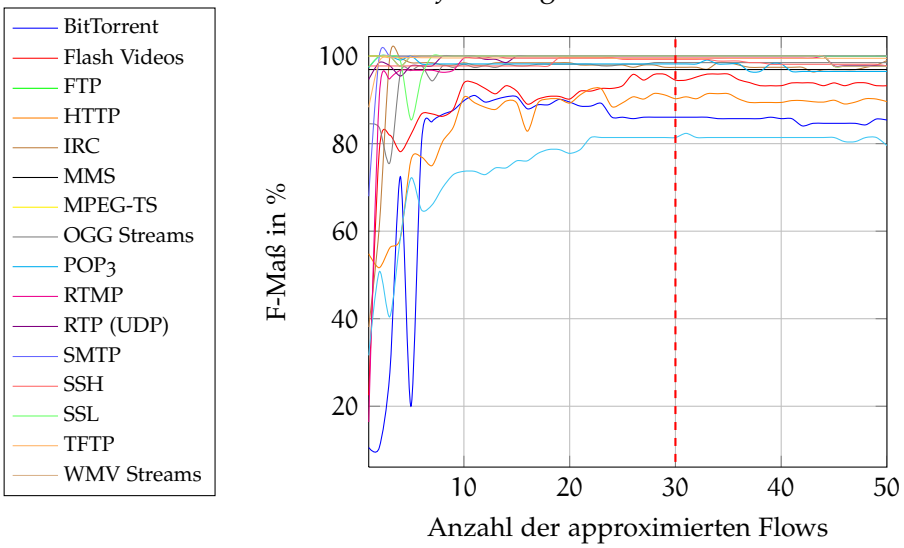

Abbildung 17: Abhängigkeit der approximierten flows. Quelle: eigene Darstellung auf Basis eigener Erhebung.

4.3.2 *Ergebnisse in Abhängigkeit der untersuchten Pakete*

Um eine möglichst schnelle Identifizierung zu erreichen wurden die Ergebnisse in Abhängigkeit der Anzahl der untersuchten Pakete untersucht. Die Ergebnisse sind in der Abbildung 18 zu sehen, die jeweils mit 30 *flows* trainiert wurden. Die Werte für WMV- und OGG-Streams steigen drastisch nach dem 13. Paket

an. Erst ab diesem Paket ist eine Unterscheidung zwischen den beiden Protokollen möglich. Ab dem 20. Paket pendelt sich der Wert von F-Maß bei allen Protokollen ein. Der Wert ist zeitlich früh genug, um eine Priorisierung des *flows* vorzunehmen. Ähnlich wie bei den Ergebnissen zuvor ist eine stabile Erkennung bei den Klartext-Protokollen bereits nach dem 10. Paket möglich.

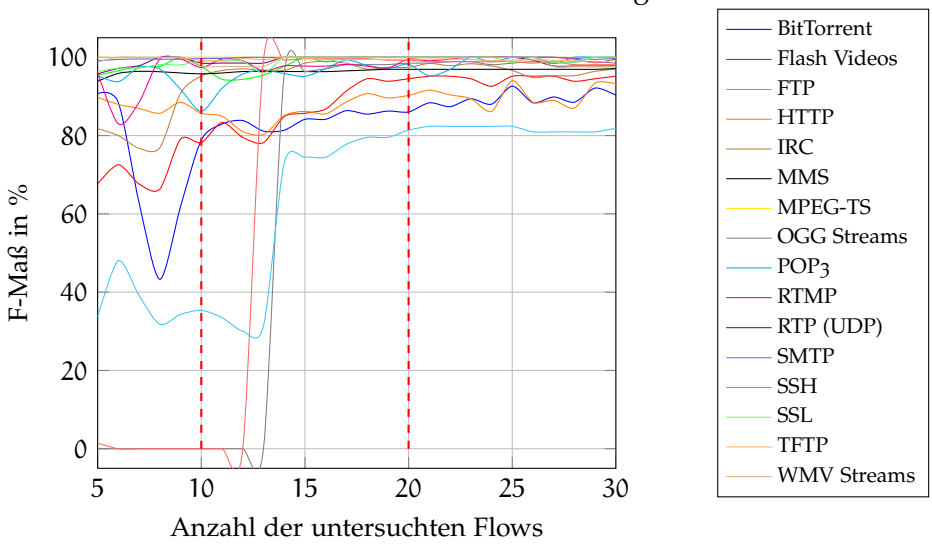

Abbildung 18: Abhängigkeit der untersuchten Pakete. Quelle: eigene Darstellung auf Basis eigener Erhebung.

4.3.3 *Ergebnisse in Abhängigkeit des Grenzwertes*

Der Grenzwert ist wie beschrieben der Wert, der entscheidet, ob ein IP-Strom erkannt wird oder nicht. Er wird verglichen mit der Summe der KLD der Messverfahren. Ist der Wert niedriger als der eingestellte Grenzwert, wird der *flow* als erkannt eingestuft und ist der Wert höher wird er als unbekannt erkannt. Die Ergebnisse in Abhängigkeit des Grenzwertes darzustellen, war wichtig, um einen optimalen Wert einzustellen. In der grafischen Darstellung 19 sind die Ergebnisse abgebildet, die mit 30 trainierten *flows* und mit 20 untersuchten Paketen gemessen wurden. Der Graph zeigt, dass der Wert 13 optimal für eine stabile Identifizierung ist, obwohl falsch-positive *flows* als unbekannt eingestuft werden.

4.3.4 *Speicherverbrauch und Geschwindigkeitstest*

4.3.4.1 *Hardware-Informationen über den Linksys WRT54GL*

Der Router *WRT54GL* ist ein Wireless Access Point der Firma Linksys. Für den Leistungstest stand dieser in der Version 1.1 zur Verfügung. Die Hardware basiert auf einer Broadcom BCM95352E CPU und ist mit 16 MB Arbeitsspeicher und 4 MB Flash-Speicher ausgestattet. Die CPU taktet mit 200 MHz und gehört der MIPS-

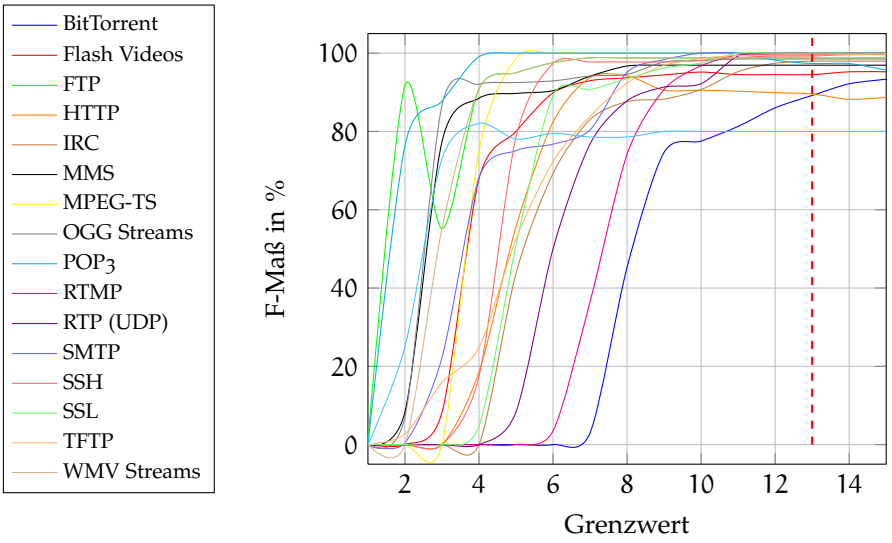

Abbildung 19: Abhängigkeit des Grenzwertes. Quelle: eigene Darstellung auf Basis eigener Erhebung.

Architektur an, die als Byte-Reihenfolge *Little-Endian* verwendet. Eine Besonderheit hat das Gerät: Es wurde nachträglich ein SD-Karten Slot angelötet, um den internen Speicher zu erweitern. Er ist in Abbildung 20 links zu sehen. Die vorinstallierte Firmware der Firma Linksys wurde durch die Linux-Distribution OpenWRT[11] 8.09 alias *Kamikaze* ersetzt, die den Linux Kernel 2.4.35.4 enthält.

4.3.4.2 *Hardware-Informationen über das IBM Thinkpad T40p*

Das IBM Thinkpad T40p hat einen Intel® Pentium® M Prozessor mit 1.6 GHz und verfügt über 1 GB an Arbeitsspeicher. Damit dieses Gerät besser mit dem Linksys WRT54GL vergleichbar ist, wurde das Notebook auf 600 MHz runter getaktet. Auf dem Gerät befindet sich ein Ubuntu 10.04 LTS mit einem 2.6.32-22 Kernel. In Abbildung 20 kann man das Gerät rechts sehen.

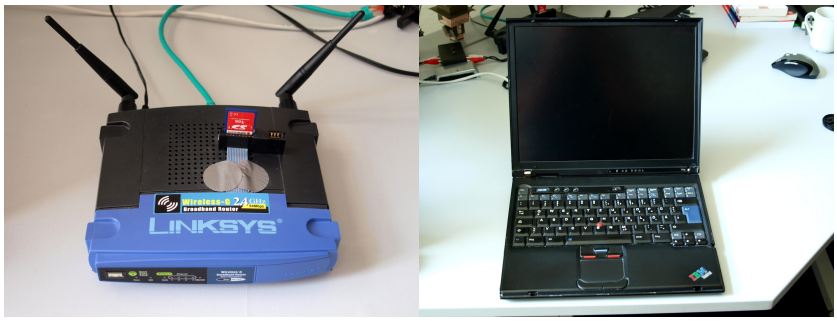

Abbildung 20: Links: Linksys WRT54GL mit SD-Karten Modifikation. Rechts: IBM Thinkpad T40p. Quelle: eigene Darstellung.

11 http://www.openwrt.org/

4.3.4.3 *Speicherverbrauch*

Den Speicherverbrauch einer Anwendung zu messen, die auf einem unixoiden Betriebssystem läuft, ist kein leichtes Unterfangen. Der Grund hierfür liegt in der Art und Weise, wie diese Systeme Programmbibliotheken laden. Programmbibliotheken sind Funktionen, Methoden oder Klassen die aus dem Programm ausgelagert sind, um von anderen Programmen verwendet werden zu können. Hierbei gibt es zwei Arten:

STATISCHE BIBLIOTHEKEN: Hierbei werden die benötigten Bibliotheken nach dem Übersetzen von dem Linker mit dem Programm verbunden. Dadurch vergrößert sich das eigentliche Programm.

DYNAMISCHE BIBLIOTHEKEN: Auch häufig unter der Bezeichnung *gemeinsam benutzte Bibliotheken* (engl. shared libraries) zu finden. Die Bibliothek wird in den Speicher geladen und kann von jedem Programm benutzt werden, welches auf sie referenziert. Die Größe des Programms verändert sich dadruch nicht.

Die meisten Bibliotheken sind dynamisch, da es Arbeitsspeicher spart und die Programme schneller laden lässt. Versucht man mit den Unix-Befehlen `ps` oder `top` den Speicherverbrauch einer Anwendung zu prüfen, so ergibt sich folgendes Problem: Der angezeigte Speicherverbrauch gilt nur, wenn die Anwendung alleine auf dem System laufen würde, da der Speicherverbrauch der dynamischen Bibliotheken mit eingerechnet wird. Vor allem Standard-Bibliotheken wie `libc` sind meist schon im Speicher und müssen daher nicht erneut geladen werden.

Das Problem zeigt, dass die Programme `top` und `ps` nicht den tatsächlichen Speicherverbrauch darstellen. Genauere Messungen sind mit dem Programm `pmap` möglich (vgl. [Devo6]). In der Auflistung 4 ist die gekürzte Ausgabe des Programms abgebildet, dass auf dem Linksys WRT54GL ausgeführt wurde.

Auflistung 4: Ausgabe des Programms pmap auf dem Linksys WRT54GL

```
root@OpenWrt:/bin# pmap -d 3422
3422:   spid -f test.db -i br-lan
Address   Kbytes Mode  Offset   Device    Mapping
00400000     200 r-x-- 00000000 079:00002 spid
00471000      20 rw--- 00031000 079:00002 spid
00476000      12 rwx-- 00000000 000:00000   [ anon ]
2aaa8000      20 r-x-- 00000000 079:00002 ld-uClibc-0.9.29.so
2aaad000       4 rw--- 00000000 000:00000   [ anon ]
2aaec000       4 r---- 00004000 079:00002 ld-uClibc-0.9.29.so
2aaed000       4 rw--- 00005000 079:00002 ld-uClibc-0.9.29.so
2aaee000     204 r-x-- 00000000 079:00002 libpcap.so.1.0.0
```

```
2ab21000     252 ----- 00033000 000:00000   [ anon ]
2ab60000       8 rw--- 00032000 079:00002 libpcap.so.1.0.0
2ab62000     528 r-x-- 00000000 079:00002 libstdc++.so.6.0.3
[...]
2acd4000       4 rw--- 0000e000 079:00002 libgcc_s.so.1
2acd5000     392 r-x-- 00000000 079:00002 libuClibc-0.9.29.so
2ad37000     252 ----- 00062000 000:00000   [ anon ]
2ad76000       4 r---- 00061000 079:00002 libuClibc-0.9.29.so
2ad77000       8 rw--- 00062000 079:00002 libuClibc-0.9.29.so
2ad79000     280 rw--- 00000000 000:00000   [ anon ]
2adbf000    3968 rw-s- 00000000 000:00003   [ anon ]
7fff6000       8 rwx-- fffff000 000:00000   [ stack ]
mapped: 7124K writeable/private: 428K shared: 3968K(*)
```

Interessant sind die drei Werte in der letzten Zeile. Der Wert 7124 KB bei *Mapped* ist der Speicherverbrauch, wenn das Programm alleine auf einem System laufen würde – ähnlich wie bei `ps` und `top`. Subtrahiert man von diesem Wert den Verbrauch der dynamischen Bibliotheken erhält man den tatsächlichen Speicherverbrauch. Diesen Wert kann man aus *writeable/private* entnehmen und beträgt: 428 KB.

Jeder Flow der im Speicher gehalten wird, nimmt bei maximaler Paketgröße folgenden zusätzlichen Speicher in Anspruch: 1500 Bytes · 20 Pakete = 30.000 Bytes $\widehat{=}$ 29,29 KB.

4.3.4.4 *Geschwindigkeitstest*

Den Versuchsaufbau kann man in Abbildung 21 sehen, der wie folgt war: Zwei Computer waren über den Router oder das Laptop verbunden, so dass jede Kommunikation zwischen den Computern A und B über die Geräte erfolgte.

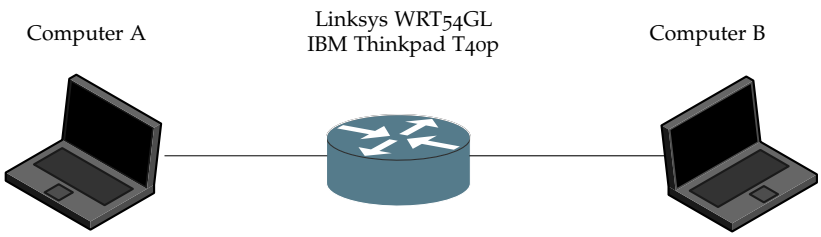

Abbildung 21: Versuchsaufbau des Geschwindigkeitstest. Quelle: [Wiko7a] (stark modifiziert).

Auf dem Linksys WRT54GL und dem IBM Thinkpad T40p lief die Implementierung von SPID, die so modifiziert wurde, dass die Zeit gemessen werden konnte. Die Zeit wird gemessen nachdem der Algorithmus die erforderliche Anzahl an Paketen gesammelt hat und umfasst die Arbeit der Messverfahren inklusive dem Vergleichen der Ergebnisse über die KLD. Die Modifikation ist in Auflistung 5 zu sehen.

Auflistung 5: Modifikation um die Zeit zu messen

```
clock_t start, end;
start = clock();
char* proto = pm->InspectFlow(tflow);
end = clock();
// [...]
unsigned int ticks = end - start;
cout << ticks / static_cast<float>(CLOCKS_PER_SEC) << " sec";
```

Die Funktion `clock()` misst die verstrichene CPU-Zeit. Die Differenz muss durch die Konstante `CLOCKS_PER_SEC` geteilt werden, um den Wert von CPU-Ticks auf Sekunden umzurechnen. Auf allen *Portable Operating System Interface (POSIX)*-kompatiblen Systemen sollte der Wert 1.000.000 betragen (vgl. [Wilo5, S. 399]). Die Implementierung wurde auf allen Plattformen mit der Option `-O3` kompiliert, die alle Optimierungen auf der Seite des Kompilierers einschaltet.

Auf Computer B lief ein simpler TCP- oder UDP-Server, der Verbindungen annahm. Computer A baute eine Verbindung zu Computer B über den Router auf und schickt jeweils 20 Pakete mit einer gewissen *Payload*-Größe. Die Größe begann bei 3 Bytes und endete bei 1460 Bytes. In Abbildung 22 sieht man die Erkennungsdauer in Abhängigkeit von der Größe der Pakete für das runter getaktete IBM Thinkpad T40p und in Abbildung 23 die entsprechenden Werte für den Linksys WRT54GL.

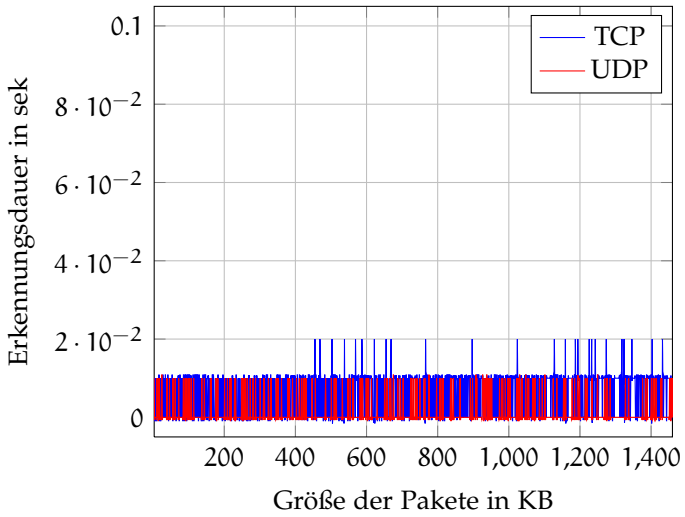

Abbildung 22: Leistungstest in Abhängigkeit der Paketgröße bei dem IBM Thinkpad T40p. Quelle: eigene Darstellung auf Basis eigener Erhebung.

Das IBM Thinkpad T40p mit 600 Mhz benötigt für UDP-*flows* zwischen 0–0,1 Sekunden und bei TCP-*flows* zwischen 0–0,2 Sekunden, obgleich welche Größe der *Payload* hat.

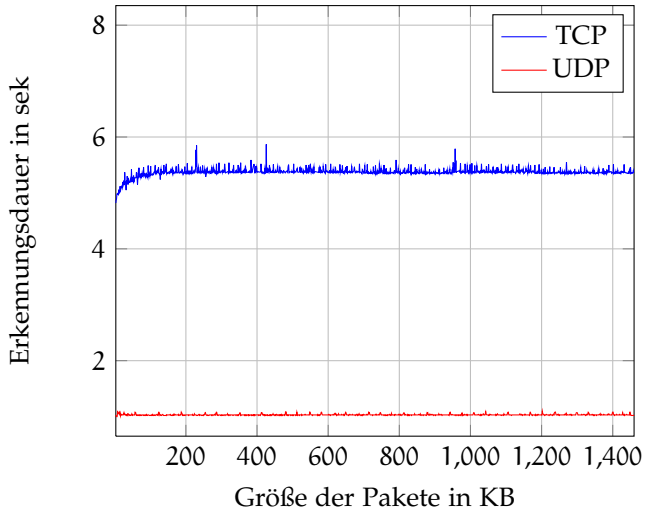

Abbildung 23: Leistungstest in Abhängigkeit der Paketgröße beim Linksys WRT54GL. Quelle: eigene Darstellung auf Basis eigener Erhebung.

Der Linksys WRT54GL mit 200 Mhz braucht für einen TCP-*Flow* zwischen 4,8–6 Sekunden und bei UDP-*Flows* zwischen 0,8–1,2 Sekunden. Bei den Ergebnissen stellt sich die Frage: Warum braucht der Router solange, um einen TCP-*Flow* zu identifizieren? Bei genauer Betrachtung findet sich die Antwort in der Abbildung 24. Dort ist die Erkennungsdauer in Abhängigkeit von der Anzahl der Protokolle in der Datenbank dargestellt, die jeweils mit 1500 Bytes großen Paketen getestet wurden.

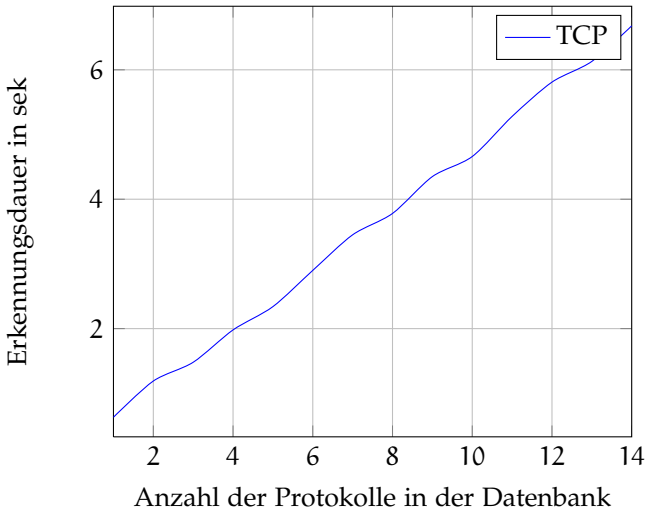

Abbildung 24: Leistungstest in Abhängigkeit der Anzahl der Protokolle in der Datenbank. Quelle: eigene Darstellung auf Basis eigener Erhebung.

Es zeigt sich, dass die Größe der Datenbank auf dem Linksys WRT54GL linear zu der Erkennungsgeschwindigkeit verläuft. Umso mehr Protokolle in der Datenbank sind, umso öfter muss

die KLD aufgerufen werden und umso mehr Rechenleistung ist erforderlich. Die vorgestellten Tests zeigen, dass der Linksys WRT54GL mit 200 MHz zu langsam für den SPID-Algorithmus ist oder nur mit einer kleinen Anzahl an ausgewählten Protokollen betrieben werden kann. Der Vergleich mit dem IBM Thinkpad T40p mit 600 MHz beweist jedoch, dass es durchaus möglich ist, für schwächere CPUs in naher-Echtzeit Protokolle zu erkennen. Für eine weitgehende Betrachtung wäre es notwendig, weitere Router zu testen wie zum Beispiel den WRT54GL in der Version 1.5, der mit 400 Mhz ausgestattet ist.

5 FAZIT

Es wurde gezeigt, dass der SPID-Algorithmus geeignet ist, um Protokolle zu erkennen und sich an der Zweckmäßigkeit der zweiten Phase von QoSiLAN richtet. Der Algorithmus erreicht sehr gute Ergebnisse für Protokolle wie Telnet, SMTP, POP3, IRC etc. liegt das F-Maß über 90 %. Die Möglichkeit die Granularität einstellen zu können, ist ein weiterer positiver Aspekt des Algorithmus. Für das Anliegen der Arbeit war es notwendig tiefer in HTTP zu schauen, um auch getunnelte Protokolle wie etwa Flash-Videos zu identifizieren. Die Evaluation zeigte, dass 30 IP-Ströme mindestens notwendig sind, um gute Ergebnisse zu erzielen. Der Geschwindigkeitstest zeigte, dass SPID in der Lage ist, auf SOHO-Geräte betrieben zu werden, die im Bereich von 200 –600 MHz liegen.

Im Vergleich zu Hjelmvik wurden mehr Protokolle inkludiert und dabei auf Streaming-Protokolle fokussiert, da diese es wert sind, priorisiert zu werden. Diese wurden im Hinblick auf ihre Genauigkeit und Robustheit evaluiert. Weiterhin wurde auf eine schnelle Implementierung geachtet, die wenig Speicherplatz belegt. Es wurde eine Auswahl an Messverfahren getroffen, die eine schnelle und präzise Erkennung ermöglichen. Zusätzlich wurde das Transport-Protokoll UDP implementiert und damit die Durchführbarkeit bewiesen. Für eine weitere Verbesserung der Implementierung ist es nötig die Leistungsmerkmale IP Version 6 und Nebenläufigkeit einzubeziehen. Summa summarum wurde gezeigt, dass der SPID-Algorithmus eine adäquate Lösung ist für die Bedingungen der zweiten Phase von QoSiLAN ist.

A

APPENDIX

PROTOKOLL	#	VAL.	TREFFERQ.	GENAUIGK.	F-MASS
BT	30	197	76.65 %	98.05 %	86.04 %
FLV	30	70	98.57 %	90.70 %	94.52 %
FTP	30	461	97.62 %	100.00 %	99.80 %
HTTP	30	127	91.34 %	88.55 %	89.92 %
IRC	30	100	95.00 %	100.00 %	98.51 %
MMS	30	252	94.05 %	100.00 %	96.93 %
MPEG-TS	30	40	100.00 %	100.00 %	100.00 %
OGG	30	122	97.06 %	100.00 %	98.51 %
POP3	30	55	100.00 %	96.49 %	98.21 %
RTMP	30	112	99.11 %	100.00 %	99.55 %
RTP/UDP	30	69	100.00 %	100.00 %	100.00 %
SMTP	30	464	100.00 %	100.00 %	100.00 %
SSH	30	136	98.53 %	100.00 %	99.26 %
SSL	30	41	100.00 %	100.00 %	100.00 %
Telnet	30	812	99.01 %	100.00 %	99.50 %
TFTP	30	42	97.62 %	100.00 %	98.80 %
WMV	30	35	100.00 %	68.63 %	81.40 %

Tabelle 5: Validierungs-Ergebnisse.

LITERATURVERZEICHNIS

[Alp04] Alpaydin, Ethem: *Maschinelles Lernen*. Oldenbourg Wissenschaftsverlag GmbH, 2004. – ISBN 978–3–486–58114–0

[Bar04] Bartsch, Hans-Jochen: *Taschenbuch Mathematischer Formeln*. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2004. – ISBN 3–446–22891–8

[Ben09] Bendrath, Ralf: Global technology trends and national regulation: Explaining Variation in the Governance of Deep Packet Inspection / Delft University of Technology. Version: Februar 2009. http://userpage.fu-berlin.de/~bendrath/ISA09_Paper_Ralf%20Bendrath_DPI.pdf, Abruf: 22. 06. 2010. 2009. – Forschungsbericht

[Bri09] Briegleb, Volker ; Heise Zeitschriften Verlag GmbH & Co KG (Hrsg.): *BT legt Pläne für umstrittenes Werbesystem auf Eis*. Version: Juli 2009. http://www.heise.de/netze/meldung/BT-legt-Plaene-fuer-umstrittenes-Werbesystem-auf-Eis-7243.html, Abruf: 18. 05. 2010

[Bun09a] Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (Hrsg.): *"Internet-Fernsehen boom" Pressemitteilung*. Version: Juni 2009. http://www.bitkom.org/de/presse/62013_59656.aspx, Abruf: 10. 06. 2010

[Bun09b] Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (Hrsg.): *"Internet-Telefonie wird immer beliebter" Pressemitteilung*. Version: August 2009. http://www.bitkom.org/60710_60706.aspx, Abruf: 10. 06. 2010

[De 05] De Montigny-Leboeuf, Annie: Flow Attributes For Use in Traffic Characterization / Communications Research Centre Canada. Version: December 2005. http://www.crc.ca/files/crc/home/research/network/system_apps/network_systems/network_security/publications/ADeMontigny_CRCTN2005003.pdf, Abruf: 22. 06. 2010. 2005. – Forschungsbericht

[Dev06] Devin: *Understanding memory usage on Linux*. Version: Februar 2006. http://virtualthreads.

blogspot.com/2006/02/understanding-memory-usage-on-linux.html, Abruf: 07. 06. 2010

[Filo6] FILDES, Jonathan: *Web inventor warns of 'dark' net*. Version: Mai 2006. http://news.bbc.co.uk/2/hi/technology/5009250.stm, Abruf: 10. 06. 2010

[Fiso8] FISCHER, Martin ; HEISE ZEITSCHRIFTEN VERLAG GMBH & CO KG (Hrsg.): *Bericht zur Ausspähung von Websurfern durch BT aufgetaucht*. Version: Juni 2008. http://www.heise.de/newsticker/meldung/Bericht-zur-Ausspaehung-von-Websurfern-durch-BT-aufgetaucht-212782.html, Abruf: 18. 05. 2010

[Frio8] FRIEDL, Jeffrey E. F.: *Reguläre Ausdrücke*. 3. O'Reilly Verlag GmbH & Co. KG, 2008. – ISBN 978–3–89721–720–1

[GRo8] GAUGELE, Jochen ; RÖTTGER, Maike ; HAMBURGER ABENDBLATT (Hrsg.): *Vorstoß: Familienministerin will Kinderpornos Im Internet sperren*. Version: November 2008. http://www.abendblatt.de/politik/deutschland/article952422/Kinderseelen-werden-zerfetzt.html, Abruf: 10. 06. 2010

[Hag10] HAGE, Simon ; MANAGER MAGAZIN ONLINE GMBH (Hrsg.): *Obermann will Google zur Kasse bitten*. Version: März 2010. http://www.manager-magazin.de/it/artikel/0,2828,684172,00.html, Abruf: 30. 03. 2010

[Hano6] HANDELSBLATT GMBH REDAKTION WIRTSCHAFTSWOCHE (Hrsg.): *Web-Riesen sollen zahlen*. Version: Februar 2006. http://www.wiwo.de/unternehmen-maerkte/web-riesen-sollen-zahlen-131752/, Abruf: 30. 03. 2010

[HJo9] HJELMVIK, Erik ; JOHN, Wolfgang: Statistical Protocol IDentification with SPID: Preliminary Results. In: *sourceSwedish National Computer Networking Workshop*, 2009

[Hjeo8] HJELMVIK, Erik: The SPID Algorithm / Swedish Internet Infrastructure Foundation. Version: Oktober 2008. http://www.iis.se/docs/The_SPID_Algorithm_-_Statistical_Protocol_IDentification.pdf, Abruf: 22. 06. 2010. 2008. – Forschungsbericht

[Hjeo9] HJELMVIK, Erik: *Wikipage of SPID*. Version: 2009. http://sourceforge.net/apps/mediawiki/spid/index.php, Abruf: 28. 03. 2010

[HL93] HAFNER, Katie ; LYON, Matthew: *Arpa Kadabra – Die Geschichte des Internet*. dpunkt - Verlag für digitale Technologie GmbH, 1993. – ISBN 3–920993–90–X

[Hur09] HURLEY, Chad: *Y,ooo,ooo,ooouTube*. Version: Oktober 2009. http://youtube-global.blogspot.com/2009/10/y000000000utube.html, Abruf: 05. 06. 2010

[KCF+08] KIM, Hyunchul ; CLAFF, K. C. ; FOMENKOV, Marina ; BARMAN, Dhiman ; FALOUTSOS, Michalis ; LEE, KiYoung: Internet traffic classification demystified: myths, caveats, and the best practices. In: *Proceedings of the 2008 ACM CoNEXT Conference*. Madrid, Spain : ACM, 2008. – ISBN 978–1–60558–210–8, S. 1–12

[Kil08] KILKKI, Kalevi: Quality of Experience in Communications Ecosystem. In: *j-jucs* 14 (2008), Nr. 5, 615–624. http://www.jucs.org/jucs_14_5/quality_of_experience_in, Abruf: 22. 06. 2010

[KL51] KULLBACK, Solomon ; LEIBLER, Richard A.: On information and sufficiency. In: *Annals of Mathematical Statistics* 22 (1951), S. 49–86

[Kle06] KLEIN, Mark: Declaration of Mark Klein in support of plaintiffs' motion for preliminary injunction / Electronic Frontier Foundation. Version: Juni 2006. http://www.eff.org/cases/att/attachments/public-unredacted-klein-declaration, Abruf: 22. 06. 2010. 2006. – Forschungsbericht

[Knu74] KNUTH, Donald E.: Computer Programming as an Art. In: *Communications of the ACM* 17 (1974), December, Nr. 12, S. 667–673

[Köh09] KÖHNEN, Christopher: QoSiLAN - A Novel QoS Strategy Using Synergy Effects of Network Topology Discovery, Traffic Analysis and NSIS QoS Signalling / City University London. 2009. – Forschungsbericht

[Kre06] KREMPL, Stefan: Der Kampf um die Netzneutralität. In: *c't – Magazin für Computertechnik* 14 (2006), 78–82. http://www.heise.de/ct/artikel/Der-Kampf-um-die-Netzneutralitaet-302658.html, Abruf: 09. 06. 2010

[Kre07] KREMPL, Stefan ; HEISE ZEITSCHRIFTEN VERLAG GMBH & CO KG (Hrsg.): *US-Kabelnetzbetreiber Comcast bremst Peer-2-Peer aus*. Version: 2007. http://www.heise.de/newsticker/meldung/US-Kabelnetzbetreiber-Comcast-bremst-Peer-2-Peer-aus-187273.html, Abruf: 28. 03. 2010

[Kre09] KREMPL, Stefan ; HEISE ZEITSCHRIFTEN VERLAG GMBH & CO KG (Hrsg.): *Bundeskabinett beschlieGesetzesentwurf zu Kinderporno-Sperren*. Version: April 2009. http://www.heise.de/newsticker/meldung/Bundeskabinett-beschliesst-Gesetzesentwurf-zu-Kinderporno-Sperren-214762.html, Abruf: 10. 06. 2010

[McC01] MCCULLAGH, Declan ; WIRED (Hrsg.): *Defcon Keeps Hackers Hooked*. Version: Juli 2001. http://web.archive.org/web/20050209031834/wired.com/news/culture/0,1284,45248-2,00.html, Abruf: 16. 06. 2010

[Mic10] MICROSOFT CORPORATION: *[MS-MMSP]: Microsoft Media Server (MMS) Protocol Specification*. Open Specifications Documentation. http://msdn.microsoft.com/en-us/library/cc234711%28PROT.10%29.aspx. Version: 2010, Abruf: 22. 06. 2010

[MJ10] MORILLON, Lucie ; JULLIARD, Jean-Fraçois: Enemies of the Internet – Countries under surveillance. In: *Reporters without Borders* (2010), März. http://www.reporter-ohne-grenzen.de/fileadmin/rte/docs/2010/Feinde_des_Internets.pdf, Abruf: 22. 06. 2010

[MP05] MOORE, Andrew W. ; PAPAGIANNAKI, Konstantina: Toward the Accurate Identification of Network Applications. In: *PAM*, 2005, S. 41–54

[MRS08] MANNING, Christopher D. ; RAGHAVAN, Prabhakar ; SCHÜTZE, Hinrich: *Introduction to Information Retrieval*. Cambridge University Press, 2008 http://nlp.stanford.edu/IR-book/information-retrieval-book.html. – ISBN 0521865719

[MW06] MADHUKAR, Alok ; WILLIAMSON, Carey: A Longitudinal Study of P2P Traffic Classification. In: *MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, IEEE Computer Society, 2006. – ISBN 0–7695–2573–3, S. 179–188

[PKB05] PRECHT, Manfred ; KRAFT, Roland ; BACHMAIER, Martin: *Angewandte Statistik 1*. Oldenbourg Wissenschaftsverlag GmbH, 2005. – ISBN 978–3486578034

[Pru09] PRUNESCU, Remus M.: *Example: Class diagram*. http://www.texample.net/tikz/examples/class-diagram/. Version: November 2009, Abruf: 22. 06. 2010. – Lizenz: CC-by-2.5

[RP94] REYNOLDS, Joyce K. ; POSTEL, Jon: *Assigned Numbers*. RFC 1700 (Historic). http://www.ietf.org/rfc/rfc1700.txt. Version: Oktober 1994, Abruf: 22. 06. 2010 (Request for Comments). – Obsoleted by RFC 3232

[RSSD04] ROUGHAN, Matthew ; SEN, Subhabrata ; SPATSCHECK, Oliver ; DUFFIELD, Nick: Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification. In: *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA : ACM, 2004. – ISBN 1–58113–821–0, S. 135–148

[SCFJ03] SCHULZRINNE, Henning ; CASNER, Stephen L. ; FREDERICK, Ron ; JACOBSON, Van: *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550 (Standard). http://www.ietf.org/rfc/rfc3550.txt. Version: Juli 2003, Abruf: 22. 06. 2010 (Request for Comments). – Updated by RFC 5506

[Sch07] SCHMEH, Klaus: *Kryptografie – Verfahren, Protokolle, Infrastruktur*. dpunkt.verlag GmbH, 2007. – ISBN 978–3–89864–435–8

[Sha48] SHANNON, Claude Elwood: A mathematical theory of communication. In: *Bell system technical journal* 27 (1948)

[SM07] SCARFONE, Karen ; MELL, Peter: Guide to Intrusion Detection and Prevention Systems (IDPS) / National Institute of Standards and Technology. Version: Februar 2007. http://csrc.ncsl.nist.gov/publications/nistpubs/800-94/SP800-94.pdf, Abruf: 22. 06. 2010. 2007. – Forschungsbericht

[Ste93] STEVENS, W. R.: *TCP/IP Illustrated Volume 1: The Protocols*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1993. – ISBN 0–201–63346–9

[Tan03] TANENBAUM, Andrew S.: *Computernetzwerke*. 4. Pearson Education Deutschland GmbH, 2003. – ISBN 3–8273–7046–9

[Vuz10] VUZE WIKI (Hrsg.): *Message Stream Encryption specification*. Version: März 2010. http://wiki.vuze.com/w/Message_Stream_Encryption, Abruf: 29. 04. 2010

[Wik07a] WIKIMEDIA: *Benutzer: George Shuklin; Datei: Router.svg*. http://commons.wikimedia.org/wiki/File:Router.svg. Version: Juni 2007, Abruf: 05. 06. 2010. – Lizenz: CC-by 2.5 Deed

[Wik07b] WIKIPEDIA DE: *Benutzer: Appaloosa; Datei: TCP Header.svg*. http://de.wikipedia.org/w/index.php?title=Datei:TCP_Header.svg. Version: July 2007, Abruf: 30. 05. 2010. – Lizenz: CC-by-sa 3.0

[Wil05] WILLEMER, Arnold: *Einstieg in C++*. Galileo Computing, 2005 http://www.willemer.de/informatik/cpp/. – ISBN 978–3–89842–649–7

[YSZ08] YU, Wang ; SHUN-ZHENG, Yu: Move Statistics-Based Traffic Classifiers Online. In: *Computer Science and Software Engineering, 2008 International Conference on* Bd. 4, 2008, S. 721–725

[ZCC00] ZWICKY, Elizabeth D. ; COOPER, Simon ; CHAPMAN, D. B.: *Building Internet Firewalls*. Bd. 2. O'Reilly & Associates, Inc., 2000. – ISBN 1–56592–871–7

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig ohne fremde Hilfe verfasst habe und keine anderen als die angegebenen Hilfsmittel von mir verwendet wurden.

Alle wörtlichen oder sinngemäßen Übernahmen aus anderen Werken wurden von mir als solche kenntlich gemacht.

Friedberg, Juni 2010

Florian E. W. Adamsky