



Masterarbeit

Erzeugung von Netzwerkverkehr mit Anomalien

Heike Austermann
Matrikelnummer: 6770231

Betreuer und Themensteller:
Prof. Dr. Jörg Keller, FernUniversität Hagen
Prof. Dr. Stefan Wendzel, Hochschule Worms

Kurzfassung

Ziel dieser Arbeit ist die Erstellung eines Tool zur Erzeugung realistisch aussehenden Netzwerkverkehrs mit festgelegten Parametern zur Verwendung beim Testen von Heuristiken. Weiterhin soll zu vorhandenem Netzwerkverkehr eine Parameterdatei erzeugt werden. Für die Umsetzung wird unter anderem das Aussehen von realistischem Netzwerkverkehrs mit Schwerpunkt auf dessen selbstähnlicher Struktur untersucht und daraus die Grundlagen für das erstellte Tool entwickelt und in diesem implementiert.

Abstract

The aim of this thesis is to create a tool for generating realistic looking network traffic with specified parameters for use in testing heuristics. Furthermore, a parameter file should be generated for existing network traffic. For the implementation the appearance of realistic network traffic with a focus on its self-similar structure is examined and the basics for the tool created are developed and implemented.

Persönliche Anmerkung zu den Fotografien

Während der Erstellung der Masterarbeit fielen immer wieder Parallelen zu den beiden großen Wassernutzungen hier um den Wohnort Clausthal im Oberharz auf. Einerseits zur historischen Nutzung im Bergbau, die unter dem Begriff „Oberharzer Wasserregal“ als Weltkulturerbe weiter gepflegt wird und deren Anlagen ähnlich einem sehr weitläufigen Freilichtmuseum erwandert werden können. Und zum zweiten zur aktuellen Nutzung für die Trinkwasserversorgung großer Teile Ostniedersachsens mit mehreren Talsperren, von denen sich die Okertalsperre und die Sösetalsperre in direkter Nachbarschaft befinden.



Abbildung 1: Ein Informationsschild zu einem Drecksumpf an der Hutthaler Widerwaage, nicht genutzt als Analogie zu Rauschen

In der ersten Version wurden die im Zusammenhang mit Netzwerkverkehr beschriebenen Ideen noch mit den Fotografien aus dem Oberharz als Erläuterung versehen. Aber auch wenn durchaus die gleichen abstrakten Prinzipien zu finden sind, so ist letztlich der Vergleich wissenschaftlich nicht stringent genug und die Anschauung zerbricht an den jeweiligen speziellen Eigenschaften. Da die Auflockierung durch Fotos jedoch den Lesefluss erleichtert, wurden einige Bilder beibehalten und als Auftakt den Kapiteln zugeordnet.

Inhaltsverzeichnis

1 Einführung und Motivation	1
1.1 Aktuelle Situation	1
1.2 Ziel dieser Masterarbeit	3
1.3 Verwandte Beiträge	3
1.4 Aufbau dieses Dokuments	4
2 Theoretischer und Technischer Hintergrund	5
2.1 Netzwerkverkehr: Definition, Eigenschaften und Abweichungsanalyse	5
2.2 Aufgezeichneter Netzwerkverkehr	7
2.3 Eigenschaften von realem Netzwerkverkehr	8
2.4 Selbstähnlichkeit, Fraktale und der Hurst-Parameter	10
2.4.1 Selbstähnlichkeit	10
2.4.2 Fraktale	12
2.4.3 H. E. Hurst und der Hurst-Parameter	13
2.4.4 Berechnungsverfahren für den Hurst-Parameter	15
2.4.5 Erzeugung einer selbstähnlichen Folge	17
2.5 Entropie	17
2.6 Anforderungsanalyse	18
3 Verwendete Verfahren	21
3.1 Gewählte Sprachen	21
3.1.1 Programmcode	22
3.1.2 Konfigurationsdatei	22
3.2 Verwendete Methoden	23
3.2.1 Zuschnitt bei Zeitplan und Modularisierung	23
3.2.2 Test-Driven Development	23
3.2.3 Versionierung	24
3.2.4 Restrukturierung	24
3.3 Genutzte Programme	25

4 Implementierung mit Erläuterungen	27
4.1 Allgemeiner Aufbau	27
4.1.1 Modul 0 Werkzeugkasten	29
4.1.2 Modul 1 Pcap-Erzeuger	29
4.1.3 Modul 2 Pcap-Ergänzer	30
4.1.4 Modul 3 Pcap-Änderer	31
4.1.5 Modul 4 Parameter-Ausleser	31
4.2 Übernommene Softwarekomponenten	32
4.2.1 aufgerufene Fremdsoftware	32
4.3 Nicht genutzte Software und Ideen	33
4.4 Vergleich und Abgrenzung	34
5 Evaluation	35
5.1 Korrektheit der Implementierung	35
5.1.1 Nutzung einer erzeugten Pcap-Datei	35
5.1.2 Parameterauswertung für einen veröffentlichten Datensatz .	36
5.1.3 Prüfung des umgesetzten Algorithmus zur Erstellung einer selbstähnlichen Folge	36
5.2 Erfahrungen zu den verwendeten Methoden	36
5.2.1 Zuschnitt bei Zeitplan und Modularisierung	36
5.2.2 Test-Driven Development	37
5.2.3 Versionierung	37
5.2.4 Restrukturierung	38
6 Zusammenfassung und Ausblick	39
6.1 Abgleich mit der Anforderungsanalyse	39
6.2 Einzelpunkte für mögliche Erweiterungen	41
6.3 Weitere Nutzung	43
Abbildungsverzeichnis	45
Literaturverzeichnis	47
Beispiel für eine Konfigurationsdatei	53
Quelltexte	55
Erklärung	57

Kapitel 1

Einführung und Motivation



Abbildung 1.1: Eingefasster Wasserlauf am Morgenbrodsthaler Graben

1.1 Aktuelle Situation

Jedes Computersystem, das auf irgendeine Weise mit anderen Systemen in Kontakt tritt, kann durch diese beeinflusst werden. Dabei können ungewollte Änderungen in den Bereichen Verfügbarkeit, Vertraulichkeit und Integrität erzwungen werden. Computersysteme benötigen daher Schutz gegen Angriffe. Dieser Schutz muss auf mehreren Ebenen erfolgen und kann sich wegen einer möglichen Kompromittierung nicht nur auf eine Beobachtung innerhalb des Computersystems beschränken. Die Ebene zur Beobachtung eines Computersystem von außerhalb besteht aus der Analyse von geändertem Verhalten. Eine gut zu untersuchende Dimension ist der Netzwerkverkehr, denn eine Änderung im Computersystem führt direkt oder in-

direkt zu abgeändertem Kommunikationsverhalten und damit Anpassungen im erzeugten Datenverkehr. Diese Anpassung kann als Anomalie, d.h. Abweichung vom normalen Verhalten, beobachtet werden. Der Vollständigkeit halber erwähnt zeigen sich im Netzwerkverkehr dabei nicht nur geänderte Kommunikationsmuster durch einen erfolgreichen Angriff sondern auch Spuren erfolgreich abgewehrter Angriffen.

Um Anomalien zeitnah erkennen und damit reagieren zu können, muss Netzwerkverkehr zügig analysiert und bewertet werden. Dazu dienen Heuristiken, deren Entwicklung und Verbesserung aktueller Forschungsgegenstand ist. Um Heuristiken zur Anomalieerkennung zu trainieren, aber auch vergleichen und damit bewerten zu können, werden Datensätze aus Netzwerkdatenverkehr genutzt. Diese Datensätze können erzeugt werden, in dem echter Datenverkehr aufgezeichnet und dieser entweder direkt genutzt wird oder vorher angepasst wird. Zusätzlich können Datensätze auch komplett synthetisch erzeugt werden. Sowohl aufgezeichneter als auch synthetisierter Netzwerkverkehr bieten je eigene Herausforderungen bei der Nutzung.

Bei aufgezeichnetem Netzwerkverkehr gibt es zwei große Probleme: Zum einen ändert sich das Aussehen von Netzwerkverkehr im Laufe der Zeit sehr stark: ein Benchmark mit aufgezeichnetem Netzwerkverkehr von vor mehreren Jahren stellt damit keinen aktuellen Netzwerkverkehr mehr dar und die Bewertung einer Heuristik an Hand dieses Datensatzes kann dementsprechend für eine aktuelle Situation nicht mehr zuverlässig getroffen werden. Ein zweites Problem aufgezeichneten Netzwerkverkehrs stellt die Anonymisierung der Daten dar, die eine sehr komplexe Nachbearbeitung insbesondere auch von allen versandten Payloads erfordert.

Bei synthetisiertem Netzwerkverkehr ist eine Vergleichbarkeit ebenfalls schwer darzustellen, da unvermeidlich beim Vorgang der Synthesierung des Netzwerkverkehrs auf eine Detektierbarkeit der eingebundenen Anomalien durch die zu bewertende Heuristik Rücksicht genommen wird. Eine weitere Herausforderung ist es, das Aussehen von echtem Netzwerkverkehr nachzustellen. Echter Netzwerkverkehr weist zwar gewisse Muster auf, diese sind aber nicht durch einfache Zufallsgenerierungen nachbildbar. Synthesizierter Netzwerkverkehr sollte nicht nur erzeugt, sondern auch in Bezug auf seine Ähnlichkeit zu echtem Netzwerkverkehr bewertet werden können.

Ein zusätzlich relevantes Problem bei der Analyse von Netzwerkverkehr ist das unvermeidbare Auftreten von Rauschen, d.h. von ungewollten und nicht analyserelevanten Objekten. Im Netzwerkverkehr kann Rauschen zum Beispiel bestehen aus fehlerhaften und entweder nicht vollständig oder auch gar nicht weiter versandten Paketen wegen eines Hardwareproblems. Rauschen kann vor einer Anomalieerkennung ausgefiltert werden, ist aber zu einem gewissen Grad in jedem (echten) Datenverkehr vorhanden und sollte daher von einer Heuristik zur Anomalieerkennung

auch korrekt interpretiert, d.h. ignoriert, werden.

Zur Zeit werden als übergreifende Vergleichsmöglichkeiten von Heuristiken zur Anomalieerkennung hauptsächlich feste und teilweise sehr alte Datensätze mit aufgezeichnetem Netzwerkverkehr genutzt.

1.2 Ziel dieser Masterarbeit

Am Abschluss dieser Arbeit soll ein Tool stehen, das synthetischen Netzwerkverkehr an Hand eines vorgegebenen Parametersatzes erzeugt, sowie aus vorhandenen Datensätzen parallele Parameter extrahiert und darüber eine übergreifende Vergleichbarkeit von Datensätzen ermöglicht.

Die individuellen Beiträge in dieser Masterarbeit zur bereits vorhandenen Landschaft sind zusammengefasst:

- Zusammenbinden der verfügbaren Tools in einer modularen, sinnvollen und getesteten Weise mit einer eindeutigen Parameterdatei als Ausgabe.
- Einfügen von Anreicherungen, die keine expliziten Angriffe darstellen, sondern Rauschen als Probleme mit der Performanz des Netzwerks wie Paketverzögerung, Leitungsausfall, etc.
- Erzeugen von und/ oder Prüfen auf und/ oder Anpassen an Selbstähnlichkeit des dargestellten Netzwerkverkehrs.

(Die Parameterextraktion wurde hier bewusst nicht als eigenen Punkt genannt, da dies auch in anderen Lösungen vorhanden ist.)

1.3 Verwandte Beiträge

Es existiert eine Vielzahl kommerzieller Tools, die Netzwerkverkehr insbesondere zum Testen von Netzwerkequipment erzeugen. Der so erzeugte Netzwerkverkehr ist allerdings nicht für eine Heuristik geeignet. Zum einen werden Lasttests erzeugt, die möglichst viel Datenverkehr in einem kurzen Zeitraum darstellen und zum anderen werden auch bewusst Fehlkonfigurationen eingebaut, um die Robustheit der eingesetzten Geräte zu prüfen.

Für die Open Source-Tools NS3 in [Mai] und OMNeT++ in [Ope] existiert bereits eine Untersuchung von Netzwerkverkehr erzeugenden Tools auf die Selbstähnlichkeit des erzeugten Netzwerkverkehrs [FNS⁺15]. Das Ergebnis zeigt jedoch, dass die untersuchten Tools die Anforderungen nur in begrenzten Fällen wie NS3 oder gar nicht wie OMNeT++ erfüllen.

1.4 Aufbau dieses Dokuments

Dieser Einleitung folgt ein Kapitel zu den theoretischen und technischen Hintergründen. In diesem Kapitel werden die oben bereits aufgeworfenen Begriffe definiert und erklärt. Es werden dabei nicht nur die technischen Protokolle und der Aufbau einer pcap-Datei dargestellt, sondern insbesondere die in der Literatur häufig vorliegende Vermischung von fraktal, selbstähnlich und dem Hurst-Parameter wird aufgetrennt. Das Kapitel schließt mit einer Anforderungsanalyse an das zu erstellende Tool ab.

Im darauf folgenden dritten Kapitel werden die bei Erstellung des Tools verwendeten Methoden genannt und erläutert. Für eine spätere Verwendbarkeit nach Abschluss dieser Masterarbeit werden dabei allgemeine Verfahren und die Zeiteinteilung kurz mitbetrachtet.

Das vierte Kapitel schildert grob den äußeren und inneren Aufbau des implementierten Tools. Zur Veranschaulichung ist auch ein Beispiel der Konfigurationsdatei im Anhang vollständig abgedruckt.

Die nachfolgende Evaluation beschreibt knapp das Vorgehen zum Nachweis der Fehlerfreiheit der Implementation und greift dann die beschriebenen Methoden wieder auf, um die Erfahrungen aus deren Nutzung darzustellen.

Den Abschluss bildet ein Abgleich mit der Anforderungsanalyse, der um mögliche Erweiterungsideen und die nächsten Schritte ergänzt wird.

Kapitel 2

Theoretischer und Technischer Hintergrund



Abbildung 2.1: Mittlerer Pfauenteich bei Clausthal

2.1 Netzwerkverkehr: Definition, Eigenschaften und Abweichungsanalyse

Ein Computernetzwerk besteht aus vielen getrennten aber miteinander verbundenen Computern, die die Rechenbedarfe einer Organisation oder einer Einheit erfüllen. Im Geschäftsbereich werden Computernetzwerke für Ressourcenaufteilung und zur Informationsteilung verwendet. Als ein Beispiel sei der einfachste Fall der Ressourcenaufteilung genannt, in dem Daten zentral auf mächtigen Com-

putern, den Servern, vorgehalten und von den Ausführenden auf simpleren Computern, den Clients, abgerufen werden. Ein Server kann hunderte oder tausende Clients gleichzeitig bedienen. Server und Clients sind über ein Netzwerk verbunden. Die Kommunikation zwischen Client und Server wird von Prozessen auf den beiden Computern realisiert, die sich gegenseitig Nachrichten über das Netzwerk senden. Eine versandte Nachrichteneinheit ist ein *Paket* und die Gesamtheit der Pakete bildet den *Netzwerkverkehr*. (Paraphrasiert nach [TW11])

Netzwerkverkehr kann zur Untersuchung aufgezeichnet werden, dies erfolgt üblicherweise nach Beschreibung einer Schnittstelle und das Ergebnis wird im zugehörigen Dateiformat mit dem Namen pcap (für packet capture) gespeichert.

Wird Netzwerkverkehr über längere Zeit beobachtet, so lassen sich bestimmte Muster als typisches Verhalten herauskristallisieren.

Diese Muster lassen sowohl qualitativ als auch quantitativ beschreiben: qualitativ werden bestimmte Services nur für bestimmte Zeitfenster und/oder bestimmte Nutzergruppen beobachtet, zum Beispiel wird eine DNS-Abfrage nur zu den in der entsprechenden Konfiguration festgelegten Servern laufen und periodische Abfragen zwischen technischen Schnittstellen nur zu bestimmten Zeiten. Quantitativ wird das Versenden großer Datenmengen über einen dafür typischen Service wie zum Beispiel FTPS oder für eine bestimmte Nutzergruppe wie zum Beispiel Updateserver beobachtet.

Mittels Heuristiken wird versucht, durch Beobachtung von Netzwerkverkehr bösartiges Verhalten durch die dadurch erzeugten Abweichungen von diesem typischen Verhalten zu identifizieren. Zum Beispiel sollte eine erhöhte Datenlast für DNS-Abfragen, unübliche periodische Anfragen an FTPS-Server mit geringer Datenlast oder eine seltene Kombination von Parametern für eine versandte Einheit auffallen.

Die zu beobachtenden Abweichungen lassen sich aufteilen in Anomalien und Rauschen. Ein Überblick findet sich in [CBK09].

Eine Anomalie ist eine bewusst herbeigeführte Abweichung vom normalen Verhalten und es ist das erklärte Ziel einer Heuristik zur Analyse von Netzwerkverkehr, diese zu identifizieren. Eine derartige Anomalie kann zum Beispiel einen Angriff mit Abfluss von Daten, einer fremdgesteuerten Verhaltensänderung eines Systems oder das Unverfügbar machen eines Service bedeuten. Im Netzwerkverkehr zeichnet sich dies durch zusätzliche, veränderte und/oder fehlende Kommunikation aus. Eine Heuristik zur Erkennung von Anomalien sollte derartige Anomalien möglichst frühzeitig erkennen können. Einen Überblick der Verfahren zur Erkennung von Anomalien gibt [BBK14]. In diesem Zusammenhang sollte vor allem beachtet werden, dass die dargestellte Liste von Datensätzen mit Netzwerkverkehr, die als Benchmark für diese Heuristiken genutzt werden, mit einem Datensatz aus 1999 beginnt und auch für jüngere Datensätze eher spärlich besetzt ist. Dieser Zustand

hat sich auch in komplett aktuellen Übersichten nicht geändert, vergleiche hierzu [RWS⁺19].

Rauschen bezeichnet ein Auftreten von ungewollten und nicht analyserelevanten Objekten. Die Änderungen durch Rauschen selber können als Ziel bei der Analyse durch eine Heuristik auffallen, sind aber an sich nicht analyserelevant und sollten auch so identifiziert werden. Im Netzwerkverkehr kann Rauschen zum Beispiel bestehen aus fehlerhaften und nicht vollständig oder gar nicht weiter versandten Paketen wegen eines Hardwareproblems. Rauschen kann vor einer Anomalieerkennung ausgefiltert werden, ist aber zu einem gewissen Grad in jedem (echten) Datenverkehr vorhanden. Rauschen sollte daher von einer Heuristik zur Anomalieerkennung auch korrekt interpretiert und ignoriert werden.

Eine nicht zu unterschätzende Schwierigkeit bietet die Nutzung von Steganographie durch einen Angreifer. Steganographie heißt, dass Informationen über nicht dafür explizit vorgesehene und damit potenziell nicht einfach von anderen identifizierbare Kanäle transportiert werden. Wird Netzwerkverkehr zur Erkennung von Anomalien beobachtet, so ist zu beachten, dass bei Ausfiltern von Rauschen Informationen verloren gehen können. Dies verhält sich analog bei einer Entfernung oder einem Angleichen von Parametern zur einfacheren Analysierbarkeit. Etwaige Angreifende versuchen natürlich einer einfachen Entdeckbarkeit ihres Vorhabens zu entgehen und erschweren das Erkennen von Anomalien dadurch, dass sie normalen Netzwerkverkehr modifizieren oder den zusätzlich erzeugten Verkehr als legitimen Netzwerkverkehr oder auch als Rauschen tarnen. Eine Heuristik zur Identifizierung von Anomalien sollte von diesem Verhalten möglichst unbeeinträchtigt bleiben. Eine vollständige Klassifikation von steganographischen Techniken findet sich in [WCM⁺21].

2.2 Aufgezeichneter Netzwerkverkehr

Netzwerkverkehr wird klassischerweise in verschiedenen OSI-Schichten aufgeteilt betrachtet. OSI stammt von der International Standards Organization (ISO) und steht für *Open Systems Interconnection*. Das so genannte OSI-Referenz-Modell besteht aus sieben Schichten, von denen jede eine andere Abstraktionsstufe und Funktion aufweist. Die sieben Schichten sind der Reihenfolge nach die physikalische, Datalink, Netzwerk, Transport, Sitzung, Präsentation und Applikation.

Packet Capture oder pcap ist eine Schnittstelle, um Netzwerkverkehr der OSI-Schichten 2 bis 7 aufzuzeichnen. Die Paketdaten werden in einer pcap-Datei gespeichert. Unter Unix wird die Bibliothek libpcap zur Aufzeichnung und Filterung der Netzwerkpakete genutzt.

Die nachfolgende Beschreibung des Formates wurde so aus der offiziellen Webpräsenz [Theb] übersetzt und zusammengefasst übernommen.

Das libpcap-Dateiformat wurde als offenes Format der „de facto“-Standard unter Unix und somit das übliche Format für Open Source. Aktuell vollzieht sich ein Umschwung zum Nachfolger, dem pcapng-Dateiformat. (Anmerkung: Umschwung steht jedoch angesichts der vielen vorliegenden Beispiele im bisherigen Format und dessen sehr guter Unterstützung noch deutlich in der ersten Hälfte.)

Die Datei selber besteht aus einem globalen Header gefolgt von null oder mehr Einträgen aufgezeichneter Pakete.

Der globale Header enthält Informationen zum Dateiformat, zur Leserichtung, zur Zeitzone, deren Genauigkeit, die Snapshot-Länge und den Datalink-Type. Beispielsweise für Ethernet ist der Datalinktype `LINKTYPE_ETHERNET` mit dem Wert 1 und für WLAN `LINKTYPE_IEEE802_11` mit dem Wert 102.

Ein Eintrag für ein aufgezeichnetes Paket besteht aus Paket-Header und Paket-Daten. Die Paket-Daten müssen nicht vollständig sein, der aufgezeichnete Eintrag enthält höchstens die ersten N Bytes jeden Paketes, die sogenannte Snapshotlänge. Diese Länge wird bei der Aufzeichnung bestimmt und kann größer gewählt werden als das größtmögliche Paket, dann typischerweise 65535, die maximale Größe für ein IP-Paket.

Der Paket-Header enthält den Zeitstempel der Aufzeichnung, die Länge des gesicherten Pakets (und somit die Anzahl der Bytes der dem Header folgenden Paket-Daten) und die ursprüngliche Länge des Paketes. Die Paketdaten folgen direkt dem Header und bestehen aus den außer der Kürzung unangepassten gelesenen Daten.

2.3 Eigenschaften von realem Netzwerkverkehr

Unmodifizierter Netzwerkverkehr weist gesamthaft betrachtet starke Schwankungen auf, die nicht gleichmäßig auf die Zeit verteilt sind. Ganz im Gegenteil zeichnet sich Netzwerkverkehr durch abrupte Spitzen hoher Aktivität aus. Diese Spitzen (in der Literatur wird dieses Muster als „burstiness“ bezeichnet) sind unabhängig vom betrachteten Zeitintervall und lassen sich auch visuell gut bestätigen.

Dem grundlegenden ersten Paper von Leland et. al. zum Aussehen von realem Netzwerkverkehr [LTWW93] ist die gezeigte Abbildung entnommen, die sowohl diese Aktivitätsspitzen als auch deren vom Maßstab unabhängiges Auftreten anschaulich zeigt.

Die Unabhängigkeit vom betrachteten Zeitintervall wird als *selbstähnlich* bezeichnet und das zugehörige Aussehen kann fraktal beschrieben werden. Das Maß der Selbstähnlichkeit wird mit dem *Hurst-Parameter* gemessen, der die Abhängigkeit der Werte in jedem Maßstab untersucht und damit ein Maß für mögliches „Aufschaukeln“ zu den beobachteten Spitzen geben kann. Anschaulich bedeutet dies: je größer die Abhängigkeit der Werte in jedem Maßstab voneinander, desto

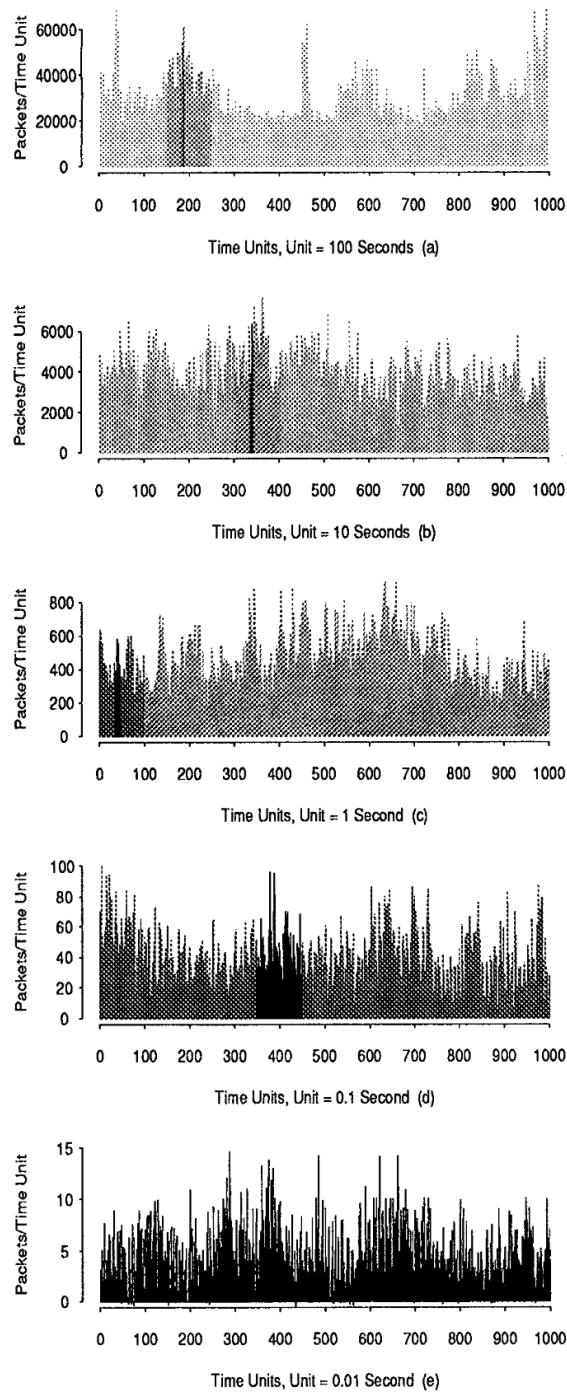


Figure 1 (a)–(e). Pictorial "proof" of self-similarity: Ethernet traffic (packets per time unit for the August '89 trace) on 5 different time scales. (Different gray levels are used to identify the same segments of traffic on the different time scales.)

Abbildung 2.2: Anschaulicher „Beweis“ für die Selbstähnlichkeit

weiter reicht der Einfluss einer Abweichung und desto stärker ist der sich ergebende Ausschlag. (Die genauen Definitionen und die Berechnung des Hurst-Parameters über die hier beschriebene Anschauung hinaus finden sich in den nachfolgenden Unterkapiteln.)

Bei der Untersuchung von Netzwerkverkehr existiert in der Literatur keine einheitliche Betrachtungsweise, ob die Paketanzahl oder der Leitungsdurchsatz pro Zeiteinheit betrachtet wird. So wird in [PF95] der Leitungsdurchsatz als Maß untersucht und führt auch (mit Ausnahmen für beschränkte Arten von Netzwerkverkehr) zu einem analogen selbstähnlichen Bild.

Ohne weitere Untersuchung gilt die Gleichsetzung von Paketanzahl und Leitungsdurchsatz anschaulich jeweils für einzelne Dienste. Wenn allerdings eher datenlastige Dienste mit leichteren Diensten gemischt werden, die auch noch ein anderes Zeitverhalten aufweisen, so ergibt sich insgesamt ein leichter Schiefstand. Anschaulich sind dies beispielsweise zum einen regelmäßige knappe (automatisierte) Abfragen versus unregelmäßig auftretende große Downloads „echter“ Nutzer.

Für die weitere Nutzung wird Netzwerkverkehr in Anlehnung an die Untersuchung in [PF95] in *nutzerbeeinflussten* und *maschinenbeeinflussten* Netzwerkverkehr aufgeteilt. (Entsprechende (übersetzte) Begrifflichkeiten wurden zwar nicht genutzt, der Netzwerkverkehr wurde aber nach der entsprechenden Logik aufgeteilt beschrieben.) Nutzerbeeinflusster Netzwerkverkehr zeigt die beschriebene burstiness unabhängig vom betrachteten Zeitintervall und ist damit selbstähnlich, in der Untersuchung betraf dies TELNET (Remotelogin) und FTP (Datentransfer). Maschinenbeeinflusster Netzwerkverkehr hingegen besteht aus regelmäßigen Abfragen nach bestimmten festen Zeitintervallen und ist daher uniform, in der Untersuchung betraf dies SMTP (Email) und NNTP (Network News).

Das Gros von realem Netzwerkverkehr ist aktuell direkt oder indirekt nutzerbeeinflusst und entsprechend weist Netzwerkverkehr insgesamt eine hohe burstiness oder Selbstähnlichkeit auf.¹

2.4 Selbstähnlichkeit, Fraktale und der Hurst-Parameter

2.4.1 Selbstähnlichkeit

Zur Erklärung von Selbstähnlichkeit folge ich für ein gutes und anschauliches Verständnis den beiden (eher) populärwissenschaftlichen Büchern [Sch94] und [Man65].

Invarianz bei Veränderung heißt allgemein, dass ein untersuchter Gegenstand

¹Inwieweit sich dieses Verhalten mit der immer weiter zunehmenden und deutlich uniformeren Maschine-zu-Maschine-Kommunikation in Zukunft ändert, ist zu beobachten.

bei einer Transformation auf sich selber abgebildet wird und es keinen Unterschied zwischen dem Original und der Transformierten gibt. Beispielsweise bildet die Rotation eines Quadrates um seinen Mittelpunkt mit Vielfachen von 90° eine Invarianz.

Selbstähnlichkeit ist allgemein eine Invarianz gegenüber Maßstabsänderungen. Diese wird auch Skaleninvarianz genannt.

Als Beispiel teilt sich ein *Flussarm* so auf, dass die Summe der Quadrate der Durchschnitte der beiden entstehenden Flussarme genau dem Quadrat des sich teilenden Flussarms entspricht. Auf der [Man65] entnommenen Grafik ist diese Aufteilung noch einmal graphisch dargestellt. Da diese Änderung die Fläche erhält, füllt der Fluss auch nach mehreren Trennungen insgesamt die Fläche in gleichem Maße aus. Durch diese Eigenschaft ist es unmöglich, bei einer Karte eines Flussdeltas mit getreu eingezeichneten Flussbreiten den Größenmaßstab anzugeben. Die Gesamtheit der Flussarme im Flussdelta ist damit selbstähnlich bzw. skaleninvariant.

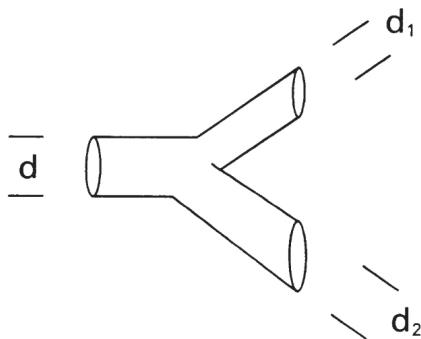


Abbildung 2.3: Aufteilung eines Flussarms

An diesem Beispiel wird auch bereits die Begrenztheit der Invarianz in der „echten“ (d.h. nicht abstrahierten) Welt sichtbar. So wird ein Flussarm mit einer sehr kleinen Breite nicht mehr im makroskopischen Gültigkeitsbereich sein, unter die Gesetze der Quantenwelt fallen und als aus einzelnen Molekülen zusammengesetzt betrachtet werden müssen. Auch der hier betrachtete Netzwerkverkehr weist klare Grenzen des Gültigkeitsbereiches auf: nach unten physikalisch durch nötigen minimalen Abstand bei Versand eines Paketes und nach oben maximal durch die Existenz von Netzwerkverkehr bzw. Computern selber.

Ein anschauliches Gegenbeispiel bildet ein *botanischer Baum*² mit einem ähnlichen Verhalten: ein Zweig teilt sich ebenfalls so in zwei Zweige auf, dass die

²Die starke Betonung auf dem Wort „botanisch“ wurde so übernommen, da in der Infomatik der Begriff Baum üblicherweise ein abstraktes Gebilde ohne definierte Breiten darstellt.

Summe der Quadrate der Durchschnitte dem Quadrat des Durchschnitts des Ursprungzweiges entspricht. Ein botanischer Baum ist jedoch nicht skaleninvariant, da die Äste nach außen schneller dünner werden als sie den Raum füllen. Wird also nur ein Teilabschnitt einer Baumkrone betrachtet, ist es sehr einfach festzustellen, an welcher Stelle und in welchem Maßstab und wie viel Anteil Baummitte und Baumaußengrenze enthalten ist, da die Äste entsprechend mehr oder weniger den Raum ausfüllen. Ein botanischer Baum ist also nicht selbstähnlich oder skaleninvariant.

Selbstähnlichkeit kann auch diskret betrachtet werden. Dann ist die Skaleninvarianz nicht kontinuierlich gegeben, sondern die Invarianz findet sich mit diskreten Skalenfaktoren.

Ein in der Informatik als beliebte Programmieraufgabe bekanntes Beispiel ist die *Koch-Kurve*. Die Koch-Kurve entsteht, indem eine Strecke in drei Teile geteilt wird und der mittlere Teil durch zwei direkt verbundene Strecken mit je einem Drittel Länge der Ursprungstrecke ersetzt wird. Jede entstandene neue Strecke wird nun wieder gedrittelt und das mittlere Drittel durch zwei weitere Strecken ersetzt. Die Koch-Kurve ist skaleninvariant zu $1/3$ bzw $1/2$. Das nachfolgende Bild ist entnommen aus [Sol].

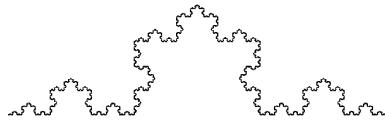


Abbildung 2.4: Koch-Kurve

2.4.2 Fraktale

In anschaulicher Definition besitzen *Fraktale* in allen Größenbereichen denselben Grad an Irregularität oder Zersplitterung. Die meisten betrachteten Fraktale sind zudem skaleninvariant.

Ein simples Beispiel für ein nicht skaleninvariantes Fraktal ist die aus drei ineinander gesetzten Koch-Kurven gebildete Schneeflocke. Das nachfolgende Bild ist entnommen aus [Arb].

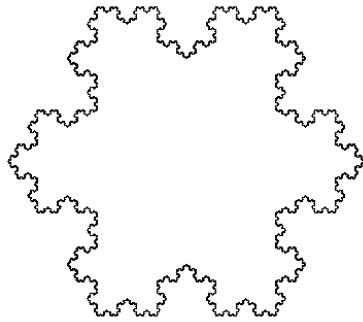


Abbildung 2.5: Schneeflocke aus Koch-Kurven

Um eine strikte Definition von Fraktalen zu erreichen, werden zunächst Längen von Kurven betrachtet und diese dann mit der Dimension in Beziehung gesetzt.

Die Länge einer Kurve kann für „glatte“ Kurven angenähert werden durch das Produkt einer festen kleinen Segmentlänge und der Anzahl gerader Segmente zu dieser Segmentlänge, die für den Lauf auf der Kurve benötigt werden. Für die genaue Länge wird dann der Grenzwert der Segmentlängen gegen Null gebildet.

Für fraktale Kurven gilt jedoch, dass die so bestimmte Länge unendlich wird, da anschaulich die Kurven so irregulär sind, dass die benötigte Anzahl von Segmenten schneller wächst als die Segmentlänge reduziert wird. Das Maß dieser Schnelligkeit ist die Hausdorff-Dimension:

$$D_H = \lim_{r \rightarrow 0} \frac{\log N}{\log(\frac{1}{r})}.$$

In mathematischer Definition sind *Fraktale* diejenigen Mengen, deren Hausdorff-Dimension echt die topologische Dimension übersteigt. Die fraktale Dimension einer glatten Kurve ist eins und für fraktale Kurven im zweidimensionalen Raum ist die Dimension echt größer als eins.

Für die Koch-Kurve kann diese Dimension anschaulich hergeleitet werden. Jede Verkleinerung der Segmentlänge um ein Drittel führt zu einer Vervierfachung der benötigten Segmente, da nun das zusätzliche Dreieck in der Mitte betrachtet werden muss und somit ergibt sich eine Dimension von

$$D_H = \frac{\log 4}{\log 3} \approx 1,26.$$

2.4.3 H. E. Hurst und der Hurst-Parameter

Der Hurst-Parameter wurde nach dem Wasserbauingenieur H. E. Hurst benannt, der unter anderem die Schwankungen in der Wassermenge des Nils untersuchte

und versuchte, die ideale Größe eines Wasserspeichers zu bestimmen, der diese ausgleichen sollte. ([SHA⁺16])

Im Nildelta wird seit Jahrtausenden der Getreideanbau durch die regelmäßigen Überschwemmungen befördert, falls diese eine optimale mittlere Höhe aufweisen. Weiterhin existieren aus vielen Zeitaltern genaue Aufzeichnungen, aus denen sich die Stände des Nils für eine rechnerische Analyse ableiten lassen. Die beiden folgenden Bilder dienen der Veranschaulichung und stammen aus einem Reisebericht zum großteils original erhaltenen Nilometer in Kairo ([Han]). Die dargestellten Einheiten sind nicht deckungsgleich, das zweite Bild zeigt die beobachtete Höhe auf der 9,5m hohen Steinsäule im Nilometer, die in insgesamt 19 Einheiten unterteilt ist.

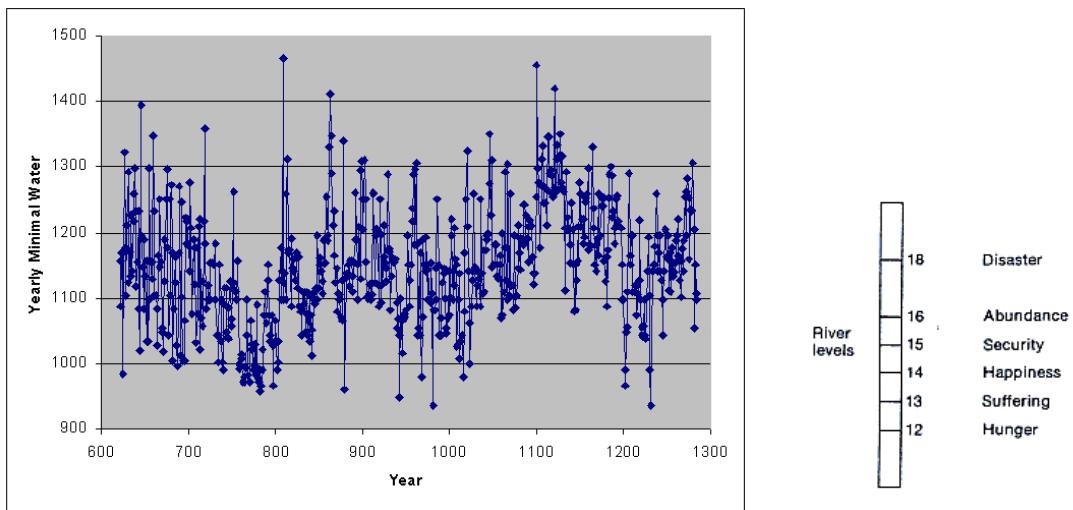


Abbildung 2.6: Nilstände aus dem frühen Mittelalter und die Auswirkungen

Ein idealer Wasserspeicher wird in jedem Jahr eine mittlere Wassermenge abgeben und einerseits so viel Wasser vorhalten, dass Jahre mit geringerer Wasserlieferung ausgeglichen werden können und andererseits so viel zusätzliche Aufnahmekapazität bieten, dass auch in Jahren mit höherer Wasserlieferung der gesamte Überschuss zurückgehalten werden kann. Anschaulich röhrt die Problematik nun daher, dass trockenere und nassere Jahre nicht unabhängig voneinander sondern üblicherweise konzentriert auftreten und somit der kumulierte Einfluss auf das Reservoir deutlich höher ist als bei einer gleichmäßigen Verteilung. Der Grad dieser Konzentration von Extremereignissen kann nun dimensionslos durch das Verhältnis vom Maximum der kumulierte Abweichungen der Kapazität zu deren Standardabweichung berechnet werden. Dieser Wert ist der Hurst-Parameter.

Der Hurst-Parameter hat Werte zwischen 0 und 1. Ein Wert größer als 0,5 weist auf einen Vorgang mit sogenannten Fernwirkungen hin, die zu dem oben so

anschaulich beschriebenen Aufschaukeln der Werte führen können. Sind die Werte unabhängig von den vorhergegangenen, so ergibt sich ein Hurst-Parameter von 0,5. Werte unterhalb von 0,5 weisen auf einen gegenteiligen Effekt hin, bei dem kleine Werte von großen gefolgt werden und umgekehrt, so dass sich insgesamt ein glatteres Bild ohne starke Verwerfungen ergibt.

Für „unangenehmere“ Flüsse mit Ballungen von Extremereignissen (wie der betrachtete Nil bei Assuan mit einem Wert von 0,9 berechnet von E. H. Hurst [Hur56] noch mit Bezeichnung K) ist der Parameter also höher als für „sanftere“ Flüsse ([Sch94] nennt als Beispiel den Rhein mit einem Wert von 0,5 für einen von Fernwirkungen unabhängigen Prozess). Allgemein ergibt sich für die von E. H. Hurst betrachteten natürlichen Phänomene in [Hur56] ein ungefährer Wert von 0,8.

Für den Zusammenhang mit der fraktalen Dimensionen gilt bei Betrachtung von Selbstähnlichkeit

$$D = E + 1 - H$$

mit D der fraktalen bzw. Hausdorff-Dimension, E der Dimension des euklidischen Raumes und H dem Hurst-Parameter.

Der Hurst-Parameter wird in der Literatur zu Netzwerkverkehr als Maßstab für die Selbstähnlichkeit genutzt, da die durch ihn beschriebene Abhängigkeit der Werte untereinander über mehrere Größenskalen hinweg anschaulich analog ist.

2.4.4 Berechnungsverfahren für den Hurst-Parameter

Die klassisch von H. E. Hurst entworfene und genutzte reskalierte Variationsbreitenanalyse (im englischen Original *rescaled range analysis*) oder auch in kurz R/S-Analyse lässt sich aus der Anwendung eines Wasserspeichers (im englischen Original *reservoir*) herleiten. Die folgende Ausführung folgt in der rechnerischen Herleitung dem Buch [Fed88].

Rechnerisch ergibt als Mittelwert $\langle \xi \rangle_\tau$ für einen Wasserfluss mit $\xi(t)$ für ein Jahr t :

$$\langle \xi \rangle_\tau = \frac{1}{\tau} \sum_{t=1}^{\tau} \xi(t)$$

Diese durchschnittliche Wassermenge wird vom Wasserspeicher in jedem Jahr abgegeben. Der kumulierte Unterschied von Wasserzufluss und Wasserabfluss ist:

$$X(t, \tau) = \sum_{u=1}^t (\xi(u) - \langle \xi \rangle_\tau).$$

Die Differenz zwischen dem Minimum und dem Maximum dieses kumulierten Unterschieds ist die Variationsbreite und bestimmt die minimale Gesamtkapazität R des Wasserspeichers:

$$R(\tau) = \max_{1 \leq t \leq \tau} X(t, \tau) - \min_{1 \leq t \leq \tau} X(t, \tau)$$

mit t der diskreten Zeiteinheit (im Beispiel das betrachtete Jahr) und 1 bis τ die betrachtete Zeitspanne (im Beispiel die Jahre mit vorliegenden Aufzeichnungen zu den Wasserständen).

Die Standardabweichung S kann aus den vorhergehenden Beobachtungen berechnet werden als:

$$S(\tau) = \left(\frac{1}{\tau} \sum_{t=1}^{\tau} (\xi(t) - \langle \xi \rangle_{\tau})^2 \right)$$

Die beobachtete reskalierte Variationsbreite R/S ist gemäß Hurst durch die folgende empirische Relation passend beschrieben:

$$R/S = (\tau/2)^H$$

mit H dem Hurst-Exponenten (von H. E. Hurst noch K genannt). Rechnerisch wird H als Steigung der Funktion der logarithmierten Gleichung bestimmt.

Die Ergebnisse nach dieser Methode führen jedoch insbesondere bei kleineren Stichproben zu sichtbaren Abweichungen. Das Berechnungsverfahren wird daher üblicherweise noch nach Anis-Lloyd um den Erwartungswert korrigiert, die Hintergründe finden sich in [AL76]. Für den Erwartungswert gilt:

$$\langle R/S \rangle_{\tau} = \begin{cases} \frac{\Gamma(\frac{\tau-1}{2})}{\sqrt{\pi}\Gamma(\frac{n}{2})} \sum_{t=1}^{\tau-1} \sqrt{\frac{\tau-t}{t}} & \tau \leq 340 \\ \frac{1}{\sqrt{n\frac{\pi}{2}}} \sum_{t=1}^{\tau-1} \sqrt{\frac{\tau-t}{t}} & \tau > 340 \end{cases}$$

Der nach Anis-Lloyd korrigierte Hurst-Exponent wird nun als $1/2$ zuzüglich der Steigung von $R/S - \langle R/S \rangle_{\tau}$ berechnet.

Anmerkung zur Auswahl bei der Implementierung: Weitere Berechnungsmethoden wie der Vergleich der Autokorrelation in den verschiedenen Skalierungsstufen nach [Dah89] sind interessant und ebenfalls anschaulich, werden aber an dieser Stelle nicht weiter betrachtet, da deren erste probeweise Implementierung keine Geschwindigkeitsvorteile ergab und auch die Berechnung keine genaueren Ergebnisse lieferte als bei Anwendung der Korrektur nach Anis-Lloyd.

2.4.5 Erzeugung einer selbstähnlichen Folge

Es gibt viele rechnerisch komplexe Modelle, eine selbstähnliche Folge zu erzeugen. Um den Rechenaufwand möglichst gering zu halten, wurde eine Verallgemeinerung eines Vorschlags aus [Sch94] umgesetzt.

Eine sehr einfache Möglichkeit mittels dreier Würfel beschreibt [Sch94] folgendermaßen: „Für eine geringe Genauigkeit genügen drei Würfel anstelle eines Zufallsprogramms als Zufallsgenerator: Der erste Würfel wird für jeden neuen Wert [...] geworfen, der zweite wird nur jedes zweite mal „aktualisiert“ und der dritte wird nur jedes vierte Mal geworfen. Diese geniale Idee stammt von Richard Voss. Die Augensumme aller drei Würfel bildet eine Zufallsvariable, die eine grobe Approximation [...] über einen begrenzten Frequenzbereich ist. In diesem spielerischen Zugang [...] werden unterschiedliche Relaxationszeiten durch unterschiedliche Liegezeiten der Würfel (in unserem Beispiel mit Faktor 2 anwachsend) gebildet.“ (ebenda, S. 139)

Diese Möglichkeit kann verallgemeinert werden auf beliebige Würfelgrößen, Vergrößerungsfaktoren und Wechselwahrscheinlichkeiten. Dass dieses Vorgehen valide ist, wird experimentell im Kapitel zur Evaluation gezeigt.

Anmerkung zur Auswahl bei der Implementierung: Weitere Berechnungsmethoden wie eine Darstellung von Netzwerkverkehr mit verschiedenen Quellen in einem ON/OFF-Wechsel nach [VL14] sind interessant, werden aber an dieser Stelle nicht weiter betrachtet, da deren erste probeweise Implementierung keine Geschwindigkeitsvorteile bei der Berechnung und keine Verbesserung der Selbstähnlichkeit gegenüber der Würfelvariante ergab.

2.5 Entropie

Ein wichtiges Kriterium bei der aktuellen Untersuchung von Netzwerkverkehr bietet die Deep Packet Inspection unter Berücksichtigung von genutzten Protokollen, Ports, Servern und vor allem des Payloads. Aktuell wird im Monitoring [Der] beim Payload vor allem die Entropie betrachtet, um indirekt aus der Informationsdichte des Payloads auf Abweichungen schließen zu können und den entsprechenden Netzwerkverkehr als potenzielle Attacke zu identifizieren.

Die Entropie in der Informationstheorie nach [Pie87] wird gemessen in bits je Symbol oder bits je Sekunde und ist definiert als die durchschnittliche Anzahl von Binärziffern, die je Symbol oder Sekunde nötig sind, um die von einer Quelle erzeugten Nachrichten zu übertragen. In der Informationstheorie wird die Entropie interpretiert als durchschnittliche Unkenntnis oder Auswahlmöglichkeit. Wählt eine Nachrichtenquelle unter n Buchstaben oder Wörtern mit von den vorhergegangenen Auswahlvorgängen unabhängigen Wahrscheinlichkeiten einen Buchstaben

oder ein Wort, so ist die Entropie definiert als

$$H = - \sum_{i=1}^n p_i \log p_i$$

bits je Buchstabe oder Wort.

Zur Eingrenzung der Berechnung der Entropie von Payloads wurde auf der diesjährigen ARES-Konferenz von Khodjaeva et. al. eine Untersuchung [KZH21] vorgestellt, ob sich bei der Detektion von DNS over HTTPS unterschiedliche Resultate ergeben, wenn alle Pakete, die ersten vier Pakete oder die ersten 96 Bytes jedes Paketes bei der Entropieberechnung beachtet werden. Vereinfacht zusammengefasst genügte eine Berechnung der ersten vier Pakete, um ein Resultat zu erzielen.

2.6 Anforderungsanalyse

Ziel dieser Arbeit ist ein Tool zur Erzeugung realistischen Netzwerkverkehrs mit festgelegten Parametern zur Verwendung beim Testen von Heuristiken. Der Netzwerkverkehr sollte im Format pcap vorliegen. Weiterhin soll zu einem vorhandenen pcap eine Parameterdatei erzeugt werden.

Realistischer Netzwerkverkehr impliziert mehr als nur ein pcap mit einer die technischen Anforderungen erfüllenden Netzwerkkommunikation. Der Inhalt selber sollte ähnlich verteilt und ähnlich aufgebaut sein wie in einem realistischen Mitschnitt.

Aus den vorhergehenden Erläuterungen folgt also, dass der erzeugte Netzwerkverkehr insgesamt eine selbstähnliche Verteilung haben sollte, die von der angesprochenen burstiness geprägt ist. Dabei ist zu beachten, dass für einzelne Kommunikationsstränge ein abweichendes uniformes Verhalten bestehen sollte, da diese eine vollständig maschinengenerierte Kommunikation darstellen.

Da fehlerfreier Netzwerkverkehr ohne Abweichungen auch nicht realistisch ist, sollten Anomalien und Rauschen angereichert werden können. Insbesondere auch für den Test von Heuristiken wird die Möglichkeit benötigt, Anomalien zu ergänzen. Um den Aufwand realistisch zu halten, sollte der Einbau von zwei Arten ausreichen. Analoges gilt für Rauschen: auch für eine fehlerbehaftete Netzwerkfunktionalität sollte ein beispielhafter Einbau von zwei Arten ausreichen.

Insgesamt ist bei der Erzeugung und Anreicherung zu beachten, dass wohlstrukturierter Netzwerkverkehr erzeugt wird, der als Grundlage für eine heuristische Prüfung dienen kann. Es sind also keine Dateien mit Protokollfehlern zu erzeugen, die eine weitere Verarbeitung unmöglich gestalten.

Die erzeugte Parameterdatei sollte vom Aussehen den üblichen Vorgaben entsprechen. Somit ist eine flache Tabelle ohne weitere Formatierungen im Format

csv vorzusehen, die aus den wichtigsten Kennwerten zu jeder Verbindung in einer Zeile besteht.

Das Tool muss nicht von Grund auf selber erstellt werden. Im Gegenteil wird davon ausgegangen, dass viele Teile aus anderen Projekten genutzt werden können. Es ist Ziel, das Tool als Ziel im akademischen Umfeld zu nutzen und daher sind dafür geeignete Lizenzen Voraussetzung.

Das Tool soll über eine Konfigurationsdatei gesteuert werden, in der sämtliche Parameter festgelegt werden. Es ist insbesondere keine graphische Oberfläche erforderlich.

Die Nutzung erfolgt unter kontrollierten Bedingungen in einer wohlmeinenden Umgebung und die Anwendung muss daher nicht speziell gegen Angriffe gehärtet werden. Selbstverständlich entbindet dies nicht von einem fehlerfreien und gut lesbaren Programmiercode mit leicht gesäuberten Eingaben.

Für die Zukunft ist vorgesehen, das Tool zu erweitern. Der Aufbau sollte daher möglichst modular erfolgen und simpel einfügbare Ergänzungen zulassen.

Kapitel 3

Verwendete Verfahren

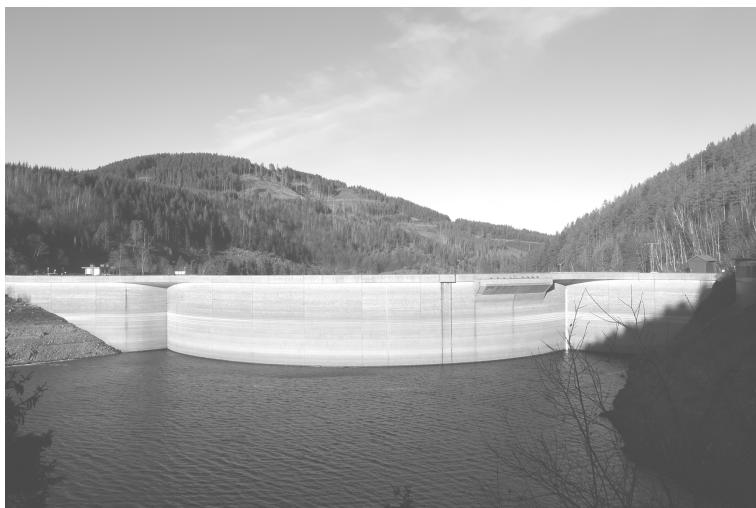


Abbildung 3.1: Sperrmauer der Okertalsperre

3.1 Gewählte Sprachen

Die Wahl einer Sprache ist zwar an sich kein Verfahren, beschränkt oder erweitert die Auswahl an verfügbaren Verfahren jedoch. Denn eine Sprache legt nicht nur die Möglichkeit für langlebigen Gebrauch, die Verfügbarkeit von Entwicklerumgebungen, die Existenz von soliden Frameworks und Bibliotheken fest sondern impliziert auch einen gewissen Grad von Eignung für gewisse Programmiermethoden und -verfahren. Das Kapitel beginnt daher mit einer kurzen Begründung der gewählten Sprachen, wobei insbesondere eine Abgrenzung zur jeweils üblichsten Alternative vorgenommen wird.

3.1.1 Programmcode

Als Programmiersprache wurde C++ gewählt. C++ wurde als Kompromiss für die Anforderungen möglichst zukunftssicher, möglichst schnell und moderne Programmiertechniken ermöglicht gewählt.

Die Zukunftssicherheit zeigt sich mit der seit über zwanzig Jahren stetigen Weiterentwicklung, einer weiten Verbreitung sowie einer angestrebte möglichst hohen Rückwärtskompatibilität.

Die Schnelligkeit ergibt sich aus der Hardwarenähe durch die Grundlegung auf dem ziemlich maschinennahen C.

Moderne Programmiertechniken lassen sich zweifellos mit C++ umsetzen. Nicht direkt durch die Sprache selber ermöglichte Konzepte werden über Frameworks oder angepasste Entwicklungsumgebungen bereitgestellt und nutzbar.

Die Alternative wäre python. Anwendungen in python sind für die meisten Fälle schnell genug und moderne Programmiertechniken werden ermöglicht. Ein Problem ist die Zukunftssicherheit. Die Sprache selber ist auf der Oberfläche einfach, aber in einer typischen Nutzung werden Frameworks per import eingebunden, die selber Abhängigkeiten vorweisen und letztlich sind die Abhängigkeiten nicht mehr gut durchblickbar und damit nicht mehr einfach beherrschbar. Insbesondere weil viele Bibliotheken noch nachgeladen werden, geht nicht nur die Sicherheit sondern auch deren eindeutiges Verhalten tendenziell zügiger verloren. Die Weiterentwicklung der Sprache selber hat mit dem Wechsel auf die Version 3 viele Inkompatibilitäten erzeugt und es ist vorauszusehen, dass die entsprechende Philosophie auch für die weitere Entwicklung bleibt.

3.1.2 Konfigurationsdatei

Für die Konfigurationsdateien wurde das Format XML gewählt.

Die offensichtliche Alternative zur Nutzung von XML für die Konfigurationsdatei ist das Herunterschreiben von mit Komma getrennten Eigenschaften in einer Textdatei. Diese Alternative bietet eine kleinere Datei und beim Lesen wären sämtliche Parameter auch in vernünftigem Rahmen in Gänze auf einer Bildschirmseite sichtbar.

Diese Nutzung setzt allerdings voraus, dass die Zeilen auch bei schnellen Änderungen und nach mehreren Monaten Nichtnutzung noch fehlerfrei interpretiert und ergänzt werden können. Dieses Leiten zu einer fehlerfreien Anpassung wird häufig durch einen Kommentar am Anfang der Datei mit Erklärungen und kurzem Beispiel sichergestellt.

Die Prüfung beim Einlesen der Konfigurationsdatei zur Verarbeitung muss beim Programmieren fast komplett selber erstellt werden und bei jeder nachfolgenden Programmerweiterung mit der Konfigurationsanpassung abgeändert werden.

Wird hingegen XML als Format für die Konfigurationsdatei genutzt, entfällt diese Notwendigkeit einer zusätzlichen Hilfestellung, denn bei Nutzung von sprechenden Eigenschaftsbezeichnungen ist die Datei selbst erklärend.

Auch das Lesen und Verstehen der Parameter durch eine dritte Person ist durch die sprechenden Umrahmungen einfacher als bei stetiger Konsultation einer Hilfestellung zum genauen Aufbau einer Datenreihe.

Weiterhin ermöglicht die Nutzung von XML die Prüfung der ordnungsgemäßen Syntax einer angepassten oder neu erstellten Konfigurationsdatei automatisiert gegen eine Definitionsdatei. Bei Programmerweiterungen mit Konfigurationsanpassung muss dann nur noch die Definitionsdatei entsprechend angepasst werden.

3.2 Verwendete Methoden

Das Kapitel enthält neben Programmiermethoden auch allgemeine Vorgehensweisen zur Arbeitslast und Zeitaufteilung. In diesem Abschnitt werden die genutzten Verfahren erläutert und mit analogem Aufbau werden im Kapitel 5 Evaluation die gemachten Erfahrungen bei der Nutzung genannt.

3.2.1 Zuschnitt bei Zeitplan und Modularisierung

Der Zeitplan für die Erstellung der Masterarbeit sah zu Beginn eine sechswöchige Phase mit Erstellung eines lauffähigen Prototypen in deutlich reduzierter Funktionalität vor. Anschließend waren die Module von Erzeugung über Ergänzung, Anpassung und schließlich zur Parametererstellung mit einer getrennten Abarbeitung im Mehrwochenrhythmus vorgesehen. Danach war eine mehrwöchige Refactoring-Phase und abschließend eine mehrwöchige Finalisierung des Thesistextes angeplant.

Der Zeitverlauf wurde großteils beibehalten, aber im Mittelteil bei Bearbeitung der Module innerhalb dieser angepasst. Einerseits wurden Module wegen gegenseitigen Abhängigkeiten parallel bearbeitet und zum anderen wurden Zeitdauern noch nachkorrigiert.

3.2.2 Test-Driven Development

Die folgende Erklärung wurde übersetzt und leicht paraphrasiert nach [Agi21b].

Test-Driven Development (TDD) ist ein Programmierstil, in dem die drei Aktivitäten Codieren, Testen (durch Schreiben von unit tests) und Designen (durch Umschreiben von Code bzw. Refactoring) verwoben sind.

Ein typischer Ablauf sieht folgendermaßen aus:

1. Einen unit test schreiben. Ein unit test ist ein Test, der einen einzelnen (neuen) Aspekt des Programms beschreibt.
2. Den Test ablaufen lassen. Dieser sollte negativ verlaufen, da der zu testende Aspekt noch nicht umgesetzt wurde. (Umgangssprachlich ist der Testfall „rot“ wegen der üblicherweise zugehörigen Farbe in der Benutzeroberfläche.)
3. Gerade genug Code schreiben, um den neuen Aspekt umzusetzen und den unit test erfolgreich positiv durchlaufen zu lassen. (Umgangssprachlich ist der Testfall „grün“ und die Vorgehensweise wird als „von rot zu grün“ umschrieben.)
4. Den Code umschreiben bis er so simpel wie möglich ist. (Der gesamte Ablauf wird auch als „rot-grün-Refactor“ bezeichnet.)
5. Den Ablauf mit dem nächsten unit test wiederholen.

3.2.3 Versionierung

Ein *Versionskontrollsystem* zeichnet Änderungen an einer Datei oder einem Satz von Dateien im Laufe der Zeit auf, damit bestimmte Versionen später abgerufen werden können. Die versionierten Dateien werden im sogenannten Repository vor gehalten. Eine Änderung wird als neue Version in das Repository eingestellt und mit einem Kommentar versehen. Bei Nutzung eines verteilten Versionskontrollsyste ms wie git kopieren die Clients das Repository einschließlich seiner vollständigen Historie.

Für die Nutzung von git als üblicherweise genutztem Versionskontrollsystem gibt es eine Vielzahl von Hilfestellungen zum Verfahren. Hervorgehoben sei an dieser Stelle [Rob12] und [Bea14]. Im ersten Beitrag werden die besten Verhal tensweisen erklärt und mit: „Stelle früh ein, perfektioniere später, veröffentliche einmal“ zusammengefasst. Der zweite Beitrag geht insbesondere auf die Kommentare (oder git commit messages) ein und hebt neben anderem die Nutzung kurzer eindeutiger Überschriften hervor.

3.2.4 Restrukturierung

Bei Programmierung des Tools wurde nach Erstellung der einzelnen Module in der Zeitplanung ein dedizierter Refactoring-Abschnitt angefügt.

Nach Definition besteht *Refactoring* darin, die interne Struktur des Quellcodes eines vorhandenen Programms zu verbessern und gleichzeitig sein externes Verhalten zu bewahren. (Übersetzt von [Agi21a].)

Bei der Zeitplanung angedacht war jedoch die umgangssprachliche Verwendung des Begriffs, der auch Codeänderungen mitmeint, die kleine zusätzliche Funktionalitäten ergänzen und auch Fehlerbehebungen enthalten. Das entsprechende Verfahren wird also passender als Restrukturierung geführt.

3.3 Genutzte Programme

Die Programmierung erfolgte unter Debian und es wurde als Entwicklungsumgebung VSCode in abschließender Version 1.60.2 genutzt. Als Testframework diente Googletest in abschließender Version 1.11.0. Zur Versionierung wurde git in Version 2.20.1 mit Server github verwendet. Als Compiler wurde gcc 8.3.0 genutzt.

Kapitel 4

Implementierung mit Erläuterungen



Abbildung 4.1: Vorstaumauer der Okertalsperre

4.1 Allgemeiner Aufbau

Das Tool besteht aus einem Verzeichnis mit der auszuführenden Datei und fünf weiteren Verzeichnissen: vier mit inhaltlich bestimmten Modulen, die komplett getrennt agieren und eines mit dem Modul für global benötigte Funktionalitäten. Es wird davon ausgegangen, dass die Fremdsoftware PcapPlusPlus in einem Ordner neben dem Verzeichnis mit dem Tool vorliegt.

Das Tool wird über eine Konfigurationsdatei gesteuert, deren Ort bei Aufruf angegeben wird. Sämtliche weiteren Einstellungen sind dann in der Konfigurati-

onsdatei in einem globalen Bereich oder in Bereichen zu den einzelnen Modulen enthalten. Beispielhaft finden sich Zuordnungen von einzelnen Adressen zu Gruppen im globalen Bereich und die Namen der erzeugten, angepassten bzw. interpretierten pcap-Dateien im Bereich zum jeweiligen Modul. Die nachfolgende Skizze soll zur Anschauung dienen.

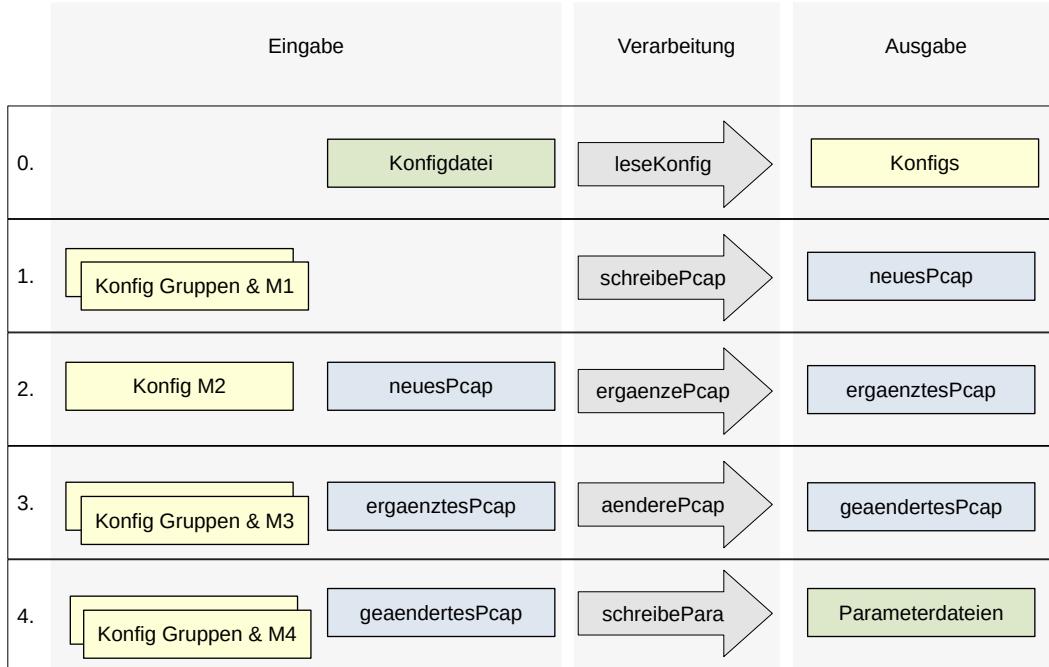


Abbildung 4.2: Skizze des inhaltlichen Programmaufbaus

Im Ausführungsverzeichnis *aufruf* befindet sich ein Makefile zur Kompilierung der ausführbaren Datei *dieApp* mit den benötigten Objekten. Danach kann das Tool mit dieser ausführbaren Datei unter Angabe der Konfigurationsdatei direkt aufgerufen werden. Im Makefile befinden sich auch Hinweise zur Bereinigung der bei einem Durchlauf erzeugten Dateien ohne bzw. mit gleichzeitiger Bereinigung um die Dateien aus der Kompilierung durch die Parameter *new* bzw. *clean*. Beispielhaft ist ein vollständiger Ablauf für zwei Aufrufe mit der Konfigurationsdatei *Konfiguration.xml* und kleiner resp. großer Bereinigung gegeben durch:

```
cd aufruf
make
./dieApp Konfiguration.xml
```

```
make new  
./dieApp Konfiguration.xml  
make clean
```

Anmerkung: bei der Anwendung sollte berücksichtigt werden, dass die Bereinigungen die erzeugten und angepassten pcap-Dateien entfernen.

Auf Grund der Anforderungen in der Anwendungsanalyse zur Nutzung unter kontrollierten Bedingungen in einer wohlmeinenden Umgebung sind keine besonderen Maßnahmen zur Härtung oder Fehlerbehandlung getroffen worden. Das Tool läuft bei Eingabe zulässiger Parameter fehlerfrei.

4.1.1 Modul 0 Werkzeugkasten

Dieses Modul enthält die gesammelten global benötigten Funktionalitäten wie den *Konfigleser*, der das Auslesen der Konfigurationsdatei realisiert, die Fremdsoftware TinyXML-2, mehrere selbst erstellte Format-Umwandlungsbibliotheken, *stringexec* für das direkte Aufrufen von Linux-Befehlen analog der Befehlszeile sowie die Sammlung der Schlüsselwörter.

Bei Umsetzung des Konfiglesers wurde von einer Expertennutzung und wie in der Anforderungsanalyse bestimmt von einer wohlwollenden Umgebung ausgegangen. Das heißt der Konfigleser führt keine gesonderten Prüfungen der Datei und ihrer Inhalte aus und gibt stets 0, den leeren String oder „undefined“ als Rückgabe, wenn eine Angabe in der Konfigurationsdatei nicht definiert wurde.

Näheres zu TinyXML-2 findet sich im Abschnitt „Übernommene Softwarekomponenten“.

Die Format-Umwandlungsbibliotheken bestehen aus *hexbytes*, das Zahlen in Zeichenketten mit den zugehörigen Hexadezimalzahlen umwandelt, sowie *removespaces*, das vom Konfigleser bei Bereinigung der Eingaben genutzt wird, um Leerzeichen zu entfernen.

Sowohl *stringexec* als auch *removespaces* hängen stark von der lokalen Implementierung ab und es wird daher zumindest für diese beiden Bibliotheken eine lokale Kompilierung empfohlen. (Aktuell ist eine vollständige lokale Kompilierung umgesetzt.)

4.1.2 Modul 1 Pcap-Erzeuger

Dieses Modul erzeugt eine pcap-Datei mit einem mitgegebenen Namen nach den in der Konfigurationsdatei ausgelesenen Parametern.

Die Objekte *Kettendef* und *Abfolge* sind reine Datencontainer. *Kettendef* enthält eine aus der Konfigurationsdatei ausgelesene Kettendefinition. Diese besteht aus den jeweiligen Gruppenzugehörigkeiten für Quelle und Ziel, dem umzusetzenden

Verfahren und ggf. zeitlichen Einschränkungen bei ihrem Auftreten. *Abfolge* enthält eine tatsächliche Realisierung einer Kette mit all ihren bei Erzeugung zufällig bestimmten Parametern (inklusive den Adressen aus den Gruppen und aus dem Verfahren sich ergebenden Ports) und dem nächsten bestimmten Zeitstempel.

Die Bibliothek *Intervalgeber* stellt mehrere Möglichkeiten bereit, eine Zufallszahl zu erhalten. Es können eine exponentielle und eine uniforme Verteilung sowie zwei fraktale Verteilungen als Grundlage ausgewählt werden. Die fraktalen Verteilungen unterscheiden sich nur darin, dass die eine einen Maximalwert beachtet.

Die Objekte *Zeitstempelgeber* dienen zur Bestimmung des nächsten Zeitstempels und werden je einer Abfolge zugeordnet. Ein Zeitstempelgeber ist entweder fraktal, dann erzeugt er eine fraktale, selbstähnliche Folge von Zeitstempeln an Hand der vorgegebenen Parameter, oder er ist nicht fraktal, dann erzeugt er eine regelmäßige Folge von Zeitstempeln mit zufälligen kleinen Abweichungen. Die beiden Arten von Zeitstempelgebern entsprechen den beiden grundsätzlich verschiedenen Verhaltensweisen von nutzerbeeinflusstem versus maschinenbeeinflusstem Datenverkehr.¹ Ein Objekt Zeitstempelgeber erzeugt den nächsten Zeitstempel in der Folge unter Nutzung der passenden Funktion des Intervallgebers, indem der gelieferte Wert als Intervall zum vorherigen Zeitstempel addiert wird.

Das Objekt *Pcapzeuger* bildet die Steuerung des Moduls. Es liest die Konfiguration aus, füllt die Randbedingungen wie maximale Paketanzahl und Zeitraum sowie die internen Vektoren mit Kettendefinitionen. Weiterhin erstellt es für den Schreibvorgang die Vektoren mit Abfolgen, den zugehörigen Zeitstempelgebern und nächsten Zeitstempeln, bestimmt die nächste zu beachtende Abfolge, sammelt deren Eigenschaften zum Erstellen des zugehörigen Pakets in die pcap-Datei, erstellt den nächsten Zeitstempel und schreibt letztlich das so festgelegte Paket in die Zielfile.

4.1.3 Modul 2 Pcap-Ergänzer

Dieses Modul ergänzt eine vorliegende pcap-Datei um Anomalien und speichert diese als ergänzte pcap-Datei.

Es besteht aus dem Objekt *PcapErgaenzer*. Bei Aufruf der *ergaenze*-Funktion werden die anzupassenden pcap-Dateien umkopiert, um die Originale zu erhalten, aus der Konfigurationsdatei die umzusetzende Funktionalität entnommen und diese schließlich aufgerufen.

¹Vergleichen Sie hierzu die Beschreibung von realem Netzwerkverkehr im Hintergrundkapitel.

4.1.4 Modul 3 Pcap-Änderer

Dieses Modul ändert eine vorliegende pcap-Datei, indem simuliertes Rauschen hinzugefügt wird und speichert diese als geänderte pcap-Datei.

Es besteht aus dem Modul *PcapAenderer* mit einem analogen äußeren Aufbau zu *PcapErgaenzer* mit Funktion *aendere*.

Die umzusetzende Änderung ergibt sich aus den folgenden Dimensionen, die aus der Konfigurationsdatei übernommen werden: Es werden

- für alle Pakete oder nur für solche mit Adressen aus bestimmten Gruppen (je für Quelle und Ziel getrennt festgelegt)
- mit bestimmter Wahrscheinlichkeit
- in einem festgelegten Zeitfenster
- Pakete gelöscht, verschieben oder gedoppelt (ggf. mit festgelegtem Intervall für Verzögerung).

Aktuell ist diese Funktionalität nur nutzbar, wenn die OSI-Schicht 2 betrachtet wird, da es sich bei den umgesetzten Rauschen-Auswirkungen um Probleme handelt, die tendenziell hardwarenäher und in dieser Schicht begründet sind.

Das Tool speichert dabei die geschobenen Pakete zwischen und fügt diese einzeln beim Durchlauf immer an der korrekten neuen Stelle in der geänderten pcap-Datei ein, so dass sich eine saubere Reihenfolge nach Zeitstempeln ergibt.

4.1.5 Modul 4 Parameter-Ausleser

Dieses Modul liest eine vorhandene pcap-Datei aus und erzeugt eine tabellarisch zusammengefasste Übersicht der einzelnen Verbindungen und/ oder eine Datei mit globalem Hurstparameter.

Das Objekt *Kette* dient als Container für eine ausgelesene Kette.² Eine Kette ist bestimmt als die Menge aller Pakete in der pcap-Datei, die dieselbe Adressen mit denselben Ports aufweisen (Quelle und Ziel können natürlich vertauscht sein; die Pakete werden dann in einer Kette zusammengefasst aber für getrennte Richtungen gezählt). Wenn der Parameter *connection assume closed after* in der Konfigurationsdatei auf einen positiven Wert gesetzt ist, wird eine Kette in zwei Ketten aufgeteilt, sobald eine Pause zwischen einzelnen Paketen vorliegt, die größer oder gleich dem festgelegten Wert ist.

²Der Begriff Kette wurde bewusst gewählt, da dieser Begriff in vorliegendem Kontext nicht mit einer eigenen Bedeutung belegt ist und somit nicht weitere Eigenschaften unterstellt werden, wie dies beispielsweise bei Verwendung des Begriffs „Konversation“ wäre.

Die beiden Bibliotheken *Entropieberechner* und *Hurstberechner* stellen die Funktionen zur Bestimmung der Entropie für einen Hexzahl-gefüllten String und des Hurst-Parameters für eine Zahlenfolge in den beiden Verfahren klassisch nach Hurst und korrigiert nach Anis-Lloyd bereit. (Die Grundlage der Rechenverfahren ist im Hintergrundkapitel beschrieben.)

Die tabellarisch zusammengefasste Übersicht mit den Kennzahlen enthält neben grundlegenden Eigenschaften bei vorheriger Berechnung die Hurst-Parameter für die Paketanzahl und den Leistungsdurchsatz sowie die Entropie der Payloads einer Kette. Die Beschränkung auf die grundlegenden Eigenschaften wie Quelle, Ziel, Zeitraum, Paketanzahl und Datenmenge wurde umgesetzt, da es bereits eine Vielzahl von Programmen zur Parametererstellung gibt.

4.2 Übernommene Softwarekomponenten

Zwei C++-Bibliotheken werden direkt im Programm genutzt. Dabei handelt es sich um TinyXML-2 zum Auslesen der Konfigurationsdatei und PcapPlusPlus für das Auslesen und Schreiben der pcap-Dateien.

Beide Bibliotheken sind open source, stehen auf github zur Verfügung und haben eine Lizenz, die eine Nutzung in diesem Kontext zulässt.

TinyXML-2 steht unter der „zlib“-Lizenz, die auch eine kommerzielle Verwendung als Teil eines Programms zulässt, solange der Quellcode mit allen Hinweisen zum Ursprung unverändert erhalten bleibt oder Änderungen eindeutig gekennzeichnet werden.

PcapPlusPlus wird unter einer so betitelten „Unlicense“ veröffentlicht, die keine Bedingungen stellt und explizit nur jede Gewährleistung ausschließt.

4.2.1 aufgerufene Fremdsoftware

Die beiden Programme id2t und pcapstego werden über im Programm eingebundene Befehlszeilen über die Kommandozeile der Linux-Umgebung aufgerufen. Beide Programme können in Modul 2 aufgerufen werden und ergänzen ein pcap um Anomalien. Es werden die Standardinstallationsparameter vorausgesetzt. Die entsprechenden Strings für den Aufruf lassen sich ggf. im Quelltext anpassen.

Die Verwendung einer indirekten Einbindung über den Aufruf per Befehlszeile und kein direktes Einbinden als Komponente ist so gewählt, weil beide Programme keine explizite API für einen Zugriff angeboten haben und zum zweiten relativ jung sind und sich damit potenziell noch stark ändern können. Eine Einbindung über einen indirekten Aufruf ist somit sowohl zukunftssicherer (der Befehl zum Aufruf ändert sich üblicherweise nicht mehr oder nur selten) als auch einfacher anzupassen (bei externem Aufruf wird nur genau diese eine Zeile angepasst).

pcapstego als Software findet sich auf github [ZC] und wird in [ZC21] genauer beschrieben. Die Software ergänzt eine vorhandene pcap-Datei um einen steganographischen Angriff. Es gibt keine Einschränkung in der Nutzung durch eine Lizenz.

ID2T als Software findet sich auf github [Tel] und wird in [VCMM16] genauer beschrieben. Die Software kann mehr als nur die vom angesprochene Ergänzung um Malware und weist auch einen dedizierten Statistikteil auf, mit dem eine pcap-Datei in ihren Paketeeigenschaften ausgewertet werden kann. Die Software steht unter der „MIT“-Lizenz, die nur das Erhalten von Copyright und Lizenzhinweisen bei Übernahme von fast allem Code bzw. mit nur geringen Anpassungen erfordert.

4.3 Nicht genutzte Software und Ideen

In den ersten Recherchen zur Nutzung angedacht aber letztlich verworfen wurden die folgenden Bibliotheken, Programme und Ideen:

- Die Software OSNT in [Net] und deren graphische Eingabemöglichkeit zur Erstellung von Einzelverbindungen wurde als unnötig erachtet, da mit der Nutzung von PcapPlusPlus jedes Paket direkt in die pcap-Datei geschrieben kann.
- Eine direkte Nutzung der Bibliothek libpcap in [Theb] wurde nicht umgesetzt. Indirekt wird libpcap jedoch als Voraussetzung für PcapPlusPlus verwendet.
- Die Bibliothek testh in [Ram] mit Hintergründen aus [Ram17] war leider auch nach mehreren Versuchen nicht lauffähig. Eine Untersuchung zur Umgehung einiger Laufprobleme wurde durch den nicht selbst erklärenden Quellcode in C erschwert. Die entsprechenden Programmteile zur Erzeugung einer selbstähnlichen Folge und zur Bestimmung des Hurst-Parameters wurden daraufhin selbst erstellt.
- Die Software INSecS-DCS wird beschrieben in [RSW18] und wurde wegen zu vieler Anforderungen an die Installation nicht betrachtet; so bestanden unter anderem Abhängigkeiten zu älteren Paketen. Eine Ergänzung für den entsprechenden Aufruf wäre zukünftig über das entsprechende Modul problemlos möglich.³

³Für das Rezept zum Einbau wird auf den Ausblick mit Skizzierung der möglichen Erweiterungen verwiesen.

- Netzwerkverkehr für einen eingeschränkten Anwendungsfälle kann durch ein geschicktes Verfahren auf einen größeren und ggf. allgemeineren Datensatz erweitert werden wie in [XMAR21] angedeutet. Eine allgemeine Übersicht zu bisherigen Untersuchungen mit Einsatz von Deep Learning zur Analyse von Netzwerkverkehr einschließlich der Vorhersage von künftigem Datenverkehr und damit der möglichen Erweiterbarkeit eines zu schmalen Datensatzes bietet [AST21]. Letztlich ist diese Möglichkeit nur ein Hilfskonstrukt, denn auch hier ist die einfache Vergleichbarkeit schwierig, da nicht alle Parameter frei wählbar sind und die genutzten Systeme im Laufe der Zeit natürlich ihr Verhalten ändern, um sich neuen Erfordernissen anzupassen. Dieses Vorgehen ist daher für die aktuelle Prüfung der Trennschärfe einer Heuristik gut geeignet, stellt aber eine übergreifende Vergleichbarkeit nicht sicher.

4.4 Vergleich und Abgrenzung

Wie in den vorhergehenden Kapiteln bereits beschrieben, ist aktuell keine Lösung bekannt, die Erzeugung, Ergänzung und Auswertung von Netzwerkverkehr in einem Tool und mit einer Konfiguration durchführt.

Viele Tools decken einen oder mehrere Teilbereich ab und sind beispielhaft in den vorherigen Kapiteln auch genannt, aber gerade im Bereich der Generierung von Netzwerkverkehr wird zum allergrößten Teil bereits vorhandener Netzwerkverkehr genutzt und angepasst.

Kapitel 5

Evaluation



Abbildung 5.1: Höhenmesser für den Wasserpegel im zum Zeitpunkt der Fotografie großteils trocken gelegten Nassewieser Teich bei Clausthal

5.1 Korrektheit der Implementierung

5.1.1 Nutzung einer erzeugten Pcap-Datei

Die erzeugten, geänderten und ergänzten pcap-Dateien sind sämtlich in wire/-shark öffnbar und auch fehlerfrei. Da jedoch Inhalte und Zähler zufällig eingestellt werden, versagt die inhaltliche Prüfung und nahezu alle Zeilen werden schwarz unterlegt dargestellt.

Eine Ausnahme besteht aktuell bei Nutzung von pcapstego: es wird ein unbekannter LinklayerTyp im Header der pcap-Datei eingestellt und wireshark stellt

damit Pakete als reine Daten dar. Die byteweise Überprüfung der einzelnen Einträge in einem Editor ergab jedoch, dass die erzeugte Datei ansonsten regulär ist und korrekt um den gewünschten steganographischen Anteil ergänzt wurde. Es wird daher angenommen, das dies nur ein temporäres Problem ist.

5.1.2 Parameterauswertung für einen veröffentlichten Datensatz

Die Erstellung der Parameter wurde neben der selbst erzeugten Datensätzen an Beispieldatensätzen teilweise aus [Theb] geprüft und beispielsweise bei [Thea] wurden die Kennzahlen korrekt bestimmt.

5.1.3 Prüfung des umgesetzten Algorithmus zur Erstellung einer selbstähnlichen Folge

Im Rahmen der automatisierten Tests wird eine Folge mit einem Hurst-Parameter von 0,72 gegen die Ergebnisse der Implementierung im Tool getestet und abschließend manuell mit den Ergebnissen für eine erzeugte pcap-Datei verglichen. So lange die Parameter so gewählt werden, dass hauptsächlich Ketten mit selbstähnlichem Aussehen erzeugt werden, ergeben sich analoge Werte.

5.2 Erfahrungen zu den verwendeten Methoden

Dieser Abschnitt enthält die persönlichen Erfahrungen zur Nutzung der Methoden, die im vorvergangenen Kapitel erklärt wurden und ist analog aufgebaut, um eine einfache Zuordenbarkeit sicherzustellen. Ich habe versucht, die Berichte wo möglich auch in der Anwendbarkeit zu verallgemeinern. Die gemachten Erfahrungen sind jedoch sehr persönlich und auf das Umfeld Masterarbeit mit entsprechender Komplexität bezogen.

5.2.1 Zuschnitt bei Zeitplan und Modularisierung

Der Beginn mit Erstellung eines Prototypen erwies sich als unverzichtbar für die eigene Motivation. Da es sich bei dem Tool um das erste sowohl selbst definierte als auch selbst geschriebene Projekt handelt, bestätigte die Fertigstellung des Prototyps die Machbarkeit.

Die weitere Aufteilung und getrennte Abarbeitung nach Modulen war für die nicht inhaltlich miteinander verwobenen Module gut. Die beiden Module zur Erstellung und zur Auswertung bedingten einander jedoch, da die Güte einer Erstellung erst nach Implementierung der Kennzahlenbestimmung bewertbar war.

Für das nächste derartige Projekt ist die Nutzung von User Stories zu empfehlen. Der Prototyp würde dann die ersten simplen Funktionalitäten abbilden, zu den unabhängigen Modulen gehörten auch jeweils eigene Stories und die inhaltliche Verwebung der Module zur Erzeugung und Auswertung wäre in korrekter Weise sichtbarer dargestellt.

5.2.2 Test-Driven Development

Das Verfahren wurde hauptsächlich am Anfang und bei Erstellung des Werkzeugkastens korrekt angewendet. Bei Umsetzung der komplizierter werdenden Algorithmen wurde entweder nur mit einfachen Durchläufen und abschließendem Prüfen der erzeugten pcap-Datei mittels wireshark, durch eine temporär eingebaute Ausgabe der relevanten Parameter oder durch manuelles Auslesen der erzeugten Parameterdatei getestet.

Für das nächste derartige Projekt sollte TDD auch während der weiteren Entwicklung im Laufe der Programmierung strikt beibehalten werden. Dies fiel in diesem Projekt mit eingeschränktem Umfang und mit den klar getrennten Modulen noch nicht auf, da der Aufbau auch im Kopf präsent sein konnte. Sobald aber etwas Zeit zwischen Entwicklungszeiten lag oder wenn das Projekt deutlich an Größe zunimmt, sollten entsprechende Testfälle für alle Funktionalitäten vorliegen.

Zwei Einzelpunkte sind des weiteren noch berichtenswert: Zum ersten kann es durch den stetigen Durchlauf des Zyklus mit Produktion von viel Codezielen zu einer Illusion von Produktivität kommen, obwohl inhaltlich noch nicht viel weiterentwickelt wurde. Zum zweiten kann es auch zwischen Testfällen zu Racingkonditionen kommen, wenn nicht jeder Testfall ein eigenes Objekt nutzt.

5.2.3 Versionierung

Die Versionierung wurde hauptsächlich als Sicherung genutzt und es wurde gerade am Anfang noch nicht viel Wert auf eindeutige kurze Beschreibungen der umgesetzten Funktionalitäten gelegt.

Für das nächste Projekt sollte gerade auf die Kurzbeschreibung geachtet werden. Zu den langen Erklärungen nach der Kurzbeschreibung kann nicht viel berichtet werden, denn diese wurden jetzt bei Nachvollziehen von Änderungen nicht genutzt, sind in einem komplexeren Projekt oder für einen Wiedereinstieg nach längerer Pause sicherlich dennoch hilfreich.

Zur Illustration der Wichtigkeit dieser Kurzbeschreibung noch beispielhaft einige tatsächlich erstellte: Wenig überraschend unterscheiden sich die Einträge „Sicherung abends“ oder „Funktion XY noch fehlerhaft“ deutlich in der späteren Nutzbarkeit schon nach mehreren Wochen auch ohne Programmierpause. Ebenfalls wird eine allgemeine Bezeichnung wie „Housekeeping“ zwar nicht gut aber

immerhin besser ersetzt durch ein leicht konkreteres „doppelter Testfall gelöscht“, „Formatierung angepasst“ oder „Namen vereinheitlicht“.

5.2.4 Restrukturierung

Eine dedizierte Zeit für die Restrukturierung zum Ende vorzusehen ist eine gute Idee. Für das nächste Projekt noch besser sind aber zusätzlich mehrere kleine Einheiten nach einer bestimmten Anzahl von User Stories einzuplanen.

Natürlich gehört das Umschreiben in den TDD-Ablauf, wurde aber selbst am Anfang als das Verfahren am Anfang noch stringenter genutzt wurde, nicht so jedes Mal umgesetzt. Und auch wenn immer mal wieder während der Programmierphasen für die Module etwas größeres angepasst wurde, war dies mit einem schlechten Gewissen verbunden, weil „Zeit für das Modul verlorengeht“. Ist dieses Umschreiben jedoch fest eingeplant, sollte diese Sorge entfallen.

Kapitel 6

Zusammenfassung und Ausblick



Abbildung 6.1: Sösetalsperre

6.1 Abgleich mit der Anforderungsanalyse

Diese Anforderungen sind erfüllt:

- Ziel dieser Arbeit ist ein Tool zur Erzeugung realistischen Netzwerkverkehrs mit festgelegten Parametern zur Verwendung beim Testen von Heuristiken. Der Netzwerkverkehr sollte im Format pcap vorliegen. Weiterhin soll zu einem vorhandenen pcap eine Parameterdatei erzeugt werden.
- Insgesamt ist bei der Erzeugung und Anreicherung zu beachten, dass wohlstrukturierter Netzwerkverkehr erzeugt wird, der als Grundlage für eine heu-

ristische Prüfung dienen kann. Es sind also keine Dateien mit Protokollfehlern zu erzeugen, die eine weitere Verarbeitung unmöglich gestalten.

- Die erzeugte Parameterdatei sollte vom Aussehen den üblichen Vorgaben entsprechen. Somit ist eine flache Tabelle ohne weitere Formatierungen im Format csv vorzusehen, die aus den wichtigsten Kenndaten zu jeder Verbindung in einer Zeile besteht.
- Das Tool muss nicht von Grund auf selber erstellt werden. Im Gegenteil wird davon ausgegangen, dass viele Teile aus anderen Projekten genutzt werden können. Es ist Ziel, das Tool als Ziel im akademischen Umfeld zu nutzen und daher sind dafür geeignete Lizenzen Voraussetzung.
- Das Tool soll über eine Konfigurationsdatei gesteuert werden, in der sämtliche Parameter festgelegt werden. Es ist insbesondere keine graphische Oberfläche erforderlich.

Diese Anforderungen sind für die beiden umgesetzten Protokollfamilien bei Nutzung realistischer Parameter erfüllt:

- Realistischer Netzwerkverkehr impliziert mehr als nur ein pcap mit einer die technischen Anforderungen erfüllenden Netzwerkkommunikation. Der Inhalt selber sollte ähnlich verteilt und ähnlich aufgebaut sein wie in einem realistischen Mitschnitt.
- Aus den vorhergehenden Erläuterungen folgt also, dass der erzeugte Netzwerkverkehr insgesamt eine selbstähnliche Verteilung haben sollte, die von der angesprochenen burstiness geprägt ist. Dabei ist zu beachten, dass für einzelne Kommunikationsstränge ein abweichendes uniformes Verhalten bestehen sollte, da diese eine vollständig maschinengenerierte Kommunikation darstellen.

Diese Anforderung ist mit den im Kapitel Implementierung dargestellten Umsetzungen erfüllt:

- Da fehlerfreier Netzwerkverkehr ohne Abweichungen auch nicht realistisch ist, sollten Anomalien und Rauschen angereichert werden können. Insbesondere auch für den Test von Heuristiken wird die Möglichkeit benötigt, Anomalien zu ergänzen. Um den Aufwand realistisch zu halten, sollte der Einbau von zwei Arten ausreichen. Analoges gilt für Rauschen: auch für eine fehlerbehaftete Netzwerkfunktionalität sollte ein beispielhafter Einbau von zwei Arten ausreichen.

Diese Anforderungen sind angestrebt, die letztendliche Erfüllung kann aber nur ein externes Review feststellen:

- Die Nutzung erfolgt unter kontrollierten Bedingungen in einer wohlmeintenden Umgebung und die Anwendung muss daher nicht speziell gegen Angriffe gehärtet werden. Selbstverständlich entbindet dies nicht von einem fehlerfreien und gut lesbaren Programmiercode mit leicht gesäuberten Eingaben.
- Für die Zukunft ist vorgesehen, das Tool zu erweitern. Der Aufbau sollte daher möglichst modular erfolgen und simpel einfügbare Ergänzungen zulassen.

Nicht in der Anforderungsanalyse genannt, aber durchaus sinnvoll sind die folgenden Anforderungen:

- Die Erstellung einer Nutzerdokumentation, die über die Beschreibungen in dieser Ausarbeitung hinausgeht und bei Veröffentlichung der Software hinzugefügt werden kann.
- Eine vollständige Abdeckung mit Tests, die insbesondere auch die Umsetzung inhaltlicher und nicht nur rein funktionale Sachverhalte prüfen.

Da die Aufgabenstellung für das Tool sehr weit gefasst ist, ist nicht nur in der Umsetzung sondern auch in der Bewertung eine Schwerpunktsetzung erfolgt und es gibt sicher viele Punkte, die auch anders hätten entschieden und umgesetzt werden können.

6.2 Einzelpunkte für mögliche Erweiterungen

Es ist in dieser Form eher unüblich, aber um die vielen möglichen Ideen, die bei Recherche und Umsetzung nicht ihren Platz in dieser Masterarbeit gefunden haben, nicht komplett in Vergessenheit geraten zu lassen, werden diese als Anschlussideen hier in einem dedizierten Kapitel einmal gesammelt dargestellt.

- Der wichtigste Punkt sind sicher Hilfetexte und Doku zur Nutzung auch außerhalb dieser Arbeit.
- Eine Ergänzung um eine DTD zur automatisierten Prüfung einer erstellten Konfigurations-Datei.
- Ergänzung um das auslaufende Protokoll ipv4, das jedoch immer noch viel genutzt wird.

- Mögliche Wahl des neuen Formats pcapng für die erzeugenden und/ oder einzulesenden Dateien.
- Echte Konversationsverläufe bei Erzeugung von Netzwerkverkehr mit Handshakes, Bestätigungen und Ende. Dazu ist die Idee jeweils eine Beispielsonversation als pcap-Datei vorzubereiten, in der nur Quelle, Ziel ersetzt werden, sowie Zähler und Payload einzeln bestimmt wird. Dies sollte flexibel genug sein, um zufällige Paketaustausche zu realisieren (also Handshakes und Ende fix, Restpakete in der Mitte werden zufällig oft erzeugt). Eine neue Verbindungsart kann dann durch Erstellen einer entsprechenden pcap-Datei und deren Nennen in der Konfigurationsdatei als Methode erfolgen und benötigt keinerlei Quellcodeanpassung.
- Eine maximale Dateigröße für eine zu erzeugende Datei und damit die Aufteilung in entsprechend mehrere Dateien bei Überschreitung.
- Ergänzung um neue Fremdapplikationen mit Aufruf über „execute“. (Dies kann am schnellsten im Quellcode umgesetzt werden über folgendes Rezept: 1. das Schlüsselwort „Ergänzungsmodus“ ergänzen, 2. den Code in Konfigleser und die zugehörigen Tests ergänzen 3. den Code in Pcapergaenzer in den switch-Statements und die zugehörigen Tests ergänzen.)
- Einbau eines alternativen Routings bei Wegfall von Verbindungen (realisiert durch Ersetzen jeweils bestimmter Adressblocks mit einem oder mehreren anderen).
- Ergänzen der Auswertung um die automatisierte Erstellung einer zip-Datei, in der die pcap-Datei mit der Parameterdatei und ggf. einem Readme oder Hilfetext zusammengefasst vorliegt.
- Zusammenfassen der Adressen nach Gruppen in den Auswertungen und dabei auch die jetzt einzelnen Berechnungen der Kennzahlen für die gesamten Gruppen durchführen.
- Komplexere Einstellungen und Erzeugungen für Payloads.
- Aktuell werden während des Hinzufügens von Rauschen Pakete beim Verschieben zwischengespeichert und an der korrekten Stelle in der pcap-Datei eingefügt. Da jedoch dafür Zwischendateien und vor allem auch deutlich mehr Aufwand nötig sind, sollte es eine Möglichkeit geben, diesen Zwischen speicher ausschaltbar zu gestalten.

6.3 Weitere Nutzung

Nach Abschluss der Masterarbeit soll der Quellcode einem Review unterzogen werden.

Nach Einarbeitung der sich daraus ergebenden Anpassungen sowie einer Ergänzung um Dokumentation und Hilfestellungen ist Ende des Jahres eine Veröffentlichung unter der MIT Lizenz auf github zum öffentlichen Abruf und insbesondere zur Weiterverwendung geplant.

Abbildungsverzeichnis

1	Ein Informationsschild zu einem Drecksumpf an der Hutthaler Widerwaage, nicht genutzt als Analogie zu Rauschen	iii
1.1	Eingefasster Wasserlauf am Morgenbrodsthaler Graben	1
2.1	Mittlerer Pfauenteich bei Clausthal	5
2.2	Anschaulicher „Beweis“ für die Selbstähnlichkeit	9
2.3	Aufteilung eines Flussarms	11
2.4	Koch-Kurve	12
2.5	Schneeflocke aus Koch-Kurven	13
2.6	Nilstände aus dem frühen Mittelalter und die Auswirkungen	14
3.1	Sperrmauer der Okertalsperre	21
4.1	Vorstaumauer der Okertalsperre	27
4.2	Skizze des inhaltlichen Programmaufbaus	28
5.1	Höhenmesser für den Wasserpegel im zum Zeitpunkt der Fotografie großteils trocken gelegten Nassewieser Teich bei Clausthal	35
6.1	Sösetalsperre	39

Literaturverzeichnis

- [Agi21a] AGILE ALLIANCE: *Refactoring*. <https://www.agilealliance.org/glossary/refactoring/>. Version: 2021. – abgerufen: 14.10.2021
- [Agi21b] AGILE ALLIANCE: *TDD*. <https://www.agilealliance.org/glossary/tdd/>. Version: 2021. – abgerufen: 14.10.2021
- [AL76] ANIS, A. A. ; LLOYD, E. H.: The expected value of the adjusted rescaled Hurst range of independent normal summands. In: *Biometrika* 63 (1976), 04, Nr. 1, S. 111–116. <http://dx.doi.org/10.1093/biomet/63.1.111>. – DOI 10.1093/biomet/63.1.111. – ISSN 0006–3444
- [Arb] ARBOL01: *Wikimedia Commons: Koch-Schneeflocke*. <https://upload.wikimedia.org/wikipedia/commons/d/d3/Flocke.png>. – abgerufen: 14.10.2021
- [AST21] ABBASI, Mahmoud ; SHAHRAKI, Amin ; TAHERKORDI, Amir: Deep Learning for Network Traffic Monitoring and Analysis (NT-MA): A Survey. In: *Computer Communications* 170 (2021), 19–41. <http://dx.doi.org/10.1016/j.comcom.2021.01.021>. – DOI 10.1016/j.comcom.2021.01.021. – ISSN 0140–3664
- [BBK14] BHUYAN, Monowar H. ; BHATTACHARYYA, D. K. ; KALITA, J. K.: Network Anomaly Detection: Methods, Systems and Tools. In: *IE-EE Communications Surveys Tutorials* 16 (2014), Nr. 1, S. 303–336. <http://dx.doi.org/10.1109/SURV.2013.052213.00046>. – DOI 10.1109/SURV.2013.052213.00046
- [Bea14] BEAMS, Chris: *How to Write a Git Commit Message*. <https://chris.beams.io/posts/git-commit/>. Version: 2014. – abgerufen: 14.10.2021
- [CBK09] CHANDOLA, Varun ; BANERJEE, Arindam ; KUMAR, Vipin: Anomaly Detection: A Survey. In: *ACM Comput. Surv.* 41 (2009), Ju-

- li, Nr. 3. <http://dx.doi.org/10.1145/1541880.1541882>. – DOI 10.1145/1541880.1541882. – ISSN 0360–0300
- [Dah89] DAHLHAUS, Rainer: Efficient Parameter Estimation for Self-Similar Processes. In: *The Annals of Statistics* 17 (1989), Nr. 4, S. 1749 – 1766. <http://dx.doi.org/10.1214/aos/1176347393>. – DOI 10.1214/aos/1176347393
- [Der] DERI, Luca: *Using nDPI for Monitoring and Security*. <https://archive.fosdem.org/2021/schedule/event/nemondpi/>. – abgerufen: 14.10.2021
- [Fed88] FEDER, Jens: *Fractals*. Plenum Press, 1988. – ISBN 0–306–42851–2
- [FNS⁺15] FERNANDES, Diogo A. ; NETO, Miguel ; SOARES, Liliana F. ; FREIRE, Mário M. ; INÁCIO, Pedro R.: Chapter 10 - On the self-similarity of traffic generated by network traffic simulators. (2015), 285–311. <http://dx.doi.org/10.1016/B978-0-12-800887-4.00010-9>. – DOI 10.1016/B978-0-12-800887-4.00010-9. ISBN 978–0–12–800887–4
- [Han] HANSEN, Roger: *The Nilometer in Cairo*. <http://www.waterhistory.org/histories/cairo/>. – abgerufen: 14.10.2021
- [Hur56] HURST, H. E.: Methods Of Using Long-Term Storage In Reservoirs. In: *Proceedings of the Institution of Civil Engineers* 5 (1956), Nr. 5, S. 519–543. <http://dx.doi.org/10.1680/iicep.1956.11503>. – DOI 10.1680/iicep.1956.11503
- [KZH21] KHODJAEVA, Yulduz ; ZINCIR-HEYWOOD, Nur: Network Flow Entropy for Identifying Malicious Behaviours in DNS Tunnels. (2021). <http://dx.doi.org/10.1145/3465481.3470089>. – DOI 10.1145/3465481.3470089. ISBN 9781450390514
- [LTWW93] LELAND, Will E. ; TAQQU, Murad S. ; WILLINGER, Walter ; WILSON, Daniel V.: On the Self-Similar Nature of Ethernet Traffic. In: *SIGCOMM Comput. Commun. Rev.* 23 (1993), Oktober, Nr. 4, S. 183—193. <http://dx.doi.org/10.1145/167954.166255>. – DOI 10.1145/167954.166255. – ISSN 0146–4833
- [Mai] MAINTAINERS: *ns-3 Network Simulator*. <https://www.nsnam.org/>. – abgerufen: 14.10.2021

- [Man65] MANDELBROT, Benoit B.: *Phänomene der Kommunikation*. Econ-Verlag GmbH, 1965. – ISBN n. angegeben
- [Net] NETFPGA: *OSNT-Public, OSNT Open Source Network Tester*. <https://github.com/NetFPGA/OSNT-Public>. – abgerufen: 14.10.2021
- [Ope] OPENSIMLTD.: *OMNeT++ Discrete Event Simulator*. <https://omnetpp.org/>. – abgerufen: 14.10.2021
- [PF95] PAXSON, V. ; FLOYD, S.: Wide area traffic: the failure of Poisson modeling. In: *IEEE/ACM Transactions on Networking* 3 (1995), Nr. 3, S. 226–244. <http://dx.doi.org/10.1109/90.392383>. – DOI 10.1109/90.392383
- [Pie87] PIERCE, John R.: *Die fraktale Geometrie der Natur*. Birkhäuser Verlag, 1987. <http://dx.doi.org/10.1007/978-3-0348-5027-8>. – ISBN 978-3-0348-5027-8
- [Ram] RAMOS, Cristiano: *TestH*. <https://github.com/cdgramos/testh>. – abgerufen: 14.10.2021
- [Ram17] RAMOS, Cristiano Duarte G.: *Improvement of TestH: a C Library for Generating Self-Similar Series and for Estimating the Hurst Parameter*. 2017 https://ubibliorum.ubi.pt/bitstream/10400.6/7836/1/5853_12365.pdf
- [Rob12] ROBERTSON, Seth: *Commit Often, Perfect Later, Publish Once: Git Best Practices*. <https://sethrobertson.github.io/GitBestPractices/>. Version: 2012. – abgerufen: 14.10.2021
- [RSW18] RAJASINGHE, Nadun ; SAMARABANDU, Jagath ; WANG, Xianbin: INSECS-DCS: A Highly Customizable Network Intrusion Dataset Creation Framework. (2018), S. 1–4. <http://dx.doi.org/10.1109/CCECE.2018.8447661>. – DOI 10.1109/CCECE.2018.8447661. – ISSN 2576–7046
- [RWS⁺19] RING, Markus ; WUNDERLICH, Sarah ; SCHEURING, Deniz ; LANDES, Dieter ; HOTHÓ, Andreas: A survey of network-based intrusion detection data sets. In: *Computers & Security* 86 (2019), 147–167. <https://www.sciencedirect.com/science/article/pii/S016740481930118X>. – ISSN 0167–4048

- [Sch94] SCHROEDER, Manfred: *Fraktale, Chaos und Selbstähnlichkeit*. Spektrum Akademischer Verlag, 1994. – ISBN 3-86025-092-2
- [SHA⁺16] SUTCLIFFE, John ; HURST, Stephen ; AWADALLAH, Ayman G. ; BROWN, Emma ; HAMED, Khaled: Harold Edwin Hurst: the Nile and Egypt, past and future. In: *Hydrological Sciences Journal* 61 (2016), Nr. 9, S. 1557–1570. <http://dx.doi.org/10.1080/02626667.2015.1019508>. – DOI 10.1080/02626667.2015.1019508
- [Sol] SOLKOLL: *Wikimedia Commons: Koch-Kurve*. https://upload.wikimedia.org/wikipedia/commons/d/d3/Koch_curve.png. – abgerufen: 14.10.2021
- [Tel] TELECOOPERATIONLAB: *ID2T - Intrusion Detection Dataset Toolkit*. <https://github.com/tklab-tud/ID2T>. – abgerufen: 14.10.2021
- [Thea] THETCPDUMP GROUP: *6LoWPAN.pcap*. <https://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=get&target=6LoWPAN.pcap.gz>. – IPv6 over IEEE 802.15.4. abgerufen: 14.10.2021
- [Theb] THETCPDUMP GROUP: *Tcpdump & Libpcap*. <https://www.tcpdump.org/>. – abgerufen: 14.10.2021
- [TW11] TANENBAUM, Andrew S. ; WETHERALL, David: *Computer networks, 5th Edition*. Pearson, 2011. – ISBN 0-13255-317-1
- [VCMM16] VASILOMANOLAKIS, Emmanouil ; CORDERO, Carlos G. ; MILANOV, Nikolay ; MÜHLHÄUSER, Max: Towards the Creation of Synthetic, yet Realistic, Intrusion Detection Datasets. (2016), S. 1209–1214. <http://dx.doi.org/10.1109/NOMS.2016.7502989>. – DOI 10.1109/NOMS.2016.7502989
- [VL14] VARET, Antoine ; LARRIEU, Nicolas: How to generate realistic network traffic ? (2014), Juli, S. pp xxxx
- [WCM⁺21] WENDZEL, Steffen ; CAVIGLIONE, Luca ; MAZURCZYK, Wojciech ; MILEVA, Aleksandra ; DITTMANN, Jana ; KRÄTZER, Christian ; LAMSHÖFT, Kevin ; VIELHAUER, Claus ; HARTMANN, Laura ; KELLER, Jörg ; NEUBERT, Tom: A Revised Taxonomy of Steganography Embedding Patterns. In: *The 16th International Conference on Availability, Reliability and Security*. New York, NY, USA : Association for Computing Machinery, 2021 (ARES 2021). – ISBN 9781450390514

- [XMAR21] XU, Shengzhe ; MARWAH, Manish ; ARLITT, Martin ; RAMAKRISHNAN, Naren: STAN: Synthetic Network Traffic Generation with Generative Neural Models. (2021)
- [ZC] ZUPPELLI, Marco ; CAVIGLIONE, Luca: *pcapStego*. https://github.com/0cram95/pcap_injector. – abgerufen: 14.10.2021
- [ZC21] ZUPPELLI, Marco ; CAVIGLIONE, Luca: PcapStego: A Tool for Generating Traffic Traces for Experimenting with Network Covert Channels. (2021). <http://dx.doi.org/10.1145/3465481.3470067>. – DOI 10.1145/3465481.3470067. ISBN 9781450390514

Konfigurationsdatei

```
<?xml version="1.0"?>
<CONFIG>
    <INCLUDELAYER2>YES</INCLUDELAYER2>
    <GROUPS>
        <GROUP><GROUPNAME>internal</GROUPNAME>
            <SRC><MAC>aa:bb:cc:dd:ee:ff</MAC><IP>1:2:3:4::8</IP></SRC>
            <SRC><MAC>a0:bb:cc:d4:e5:f6</MAC><IP>1:2:3:4::5</IP></SRC>
            <SRC><MAC>44:44:22:12:34:88</MAC><IP>1:2:3:4::ef</IP></SRC>
        </GROUP>
        <GROUP><GROUPNAME>externalspecial</GROUPNAME>
            <SRC><MAC>aa:11:bb:22:cc:33</MAC><IP>8abc:7f:6e:5d::1</IP></SRC>
        </GROUP>
        <GROUP><GROUPNAME>external</GROUPNAME>
            <SRC><MAC>00:11:22:33:44:00</MAC><IP>1::cdef</IP></SRC>
            <SRC><MAC>aa:11:bb:22:cc:33</MAC><IP>8abc:7f:6e:5d::1</IP></SRC>
        </GROUP>
    </GROUPS>
    <MODULE1>
        <EXECUTE>YES</EXECUTE>
        <NAME>neuesPcap</NAME>
        <MAXNUMBER>10000</MAXNUMBER>
        <STARTTIME>1619872968012345</STARTTIME>
        <ENDTIME>1620473684012345</ENDTIME>
        <CHAINSTARTINTERVAL>100</CHAINSTARTINTERVAL>
        <MAXCHAINS>99</MAXCHAINS>
        <CHAINS>
            <CHAIN>
                <CHAINNAME>standard</CHAINNAME>
                <WEIGHT>10</WEIGHT><LATESTART>0</LATESTART><EARLYEND>0</EARLYEND>
                <FROMGROUP>internal</FROMGROUP><TOGROUP>external</TOGROUP>
                <METHOD>APPDATA</METHOD><INTERVAL>10000</INTERVAL><FACTOR>10</FACTOR><RATIO>0.5</RATIO>
            </CHAIN>
            <CHAIN>
                <CHAINNAME>dnsprototyp</CHAINNAME>
                <WEIGHT>1</WEIGHT><LATESTART>0</LATESTART><EARLYEND>0</EARLYEND>
                <FROMGROUP>internal</FROMGROUP><TOGROUP>externalspecial</TOGROUP>
                <METHOD>DNS</METHOD><INTERVAL>300000</INTERVAL><FACTOR>100</FACTOR>
            </CHAIN>
            <CHAIN>
                <CHAINNAME>extern</CHAINNAME>
                <WEIGHT>3</WEIGHT><LATESTART>1000</LATESTART><EARLYEND>876543</EARLYEND>
                <FROMGROUP>externalspecial</FROMGROUP><TOGROUP>internal</TOGROUP>
                <METHOD>APPDATA</METHOD><INTERVAL>10000</INTERVAL><FACTOR>100</FACTOR><RATIO>0.01</RATIO>
            </CHAIN>
        </CHAINS>
    </MODULE1>
    <MODULE2>
        <EXECUTE>NO</EXECUTE>
        <NAME>ergaenztesPcap</NAME>
        <SOURCE>neuesPcap</SOURCE>
        <COMMAND>
            <METHOD>PCAPSTEGO</METHOD>
            <INFO>ressourcen/attacks.txt</INFO>
        </COMMAND>
        <COMMAND>
            <METHOD>ID2T</METHOD>
            <INFO>PortscanAttack ip.src=1::cdef mac.src=00:11:22:33:44:00 inject.at-timestamp=1619872968</INFO>
        </COMMAND>
    </MODULE2>
    <MODULE3>
        <EXECUTE>YES</EXECUTE>
        <NAME>geaendertesPcap</NAME>
        <SOURCE>neuesPcap</SOURCE>
        <FAULTYCONNECTION>
```

```

<STARTTIME>1619872968012345</STARTTIME><ENDTIME>1620473684012345</ENDTIME>
<FAULTYRATIO>0.9</FAULTYRATIO>
<FAULTYTYPE>DELETE</FAULTYTYPE><FAULTYTIMESHIFT>100</FAULTYTIMESHIFT>
<CONCERNING>FROMGROUPTOGROUP </CONCERNING>
<FROMGROUP>internal</FROMGROUP><TOGROUP>external</TOGROUP>
</FAULTYCONNECTION>
<FAULTYCONNECTION>
<STARTTIME>1619872969117191</STARTTIME><ENDTIME>1619872969728412</ENDTIME>
<FAULTYRATIO>0.5</FAULTYRATIO>
<FAULTYTYPE>COPY</FAULTYTYPE><FAULTYTIMESHIFT>100</FAULTYTIMESHIFT>
<CONCERNING>FROMGROUPTOGROUP </CONCERNING>
<FROMGROUP>internal</FROMGROUP><TOGROUP>externalspecial</TOGROUP>
</FAULTYCONNECTION>
<FAULTYCONNECTION>
<STARTTIME>1619872969117191</STARTTIME><ENDTIME>1619872969728412</ENDTIME>
<FAULTYRATIO>1</FAULTYRATIO>
<FAULTYTYPE>SHIFT</FAULTYTYPE><FAULTYTIMESHIFT>100</FAULTYTIMESHIFT>
<CONCERNING>FROMGROUPTOGROUP </CONCERNING>
<FROMGROUP>externalspecial</FROMGROUP><TOGROUP>internal</TOGROUP>
</FAULTYCONNECTION>
</MODULE3>
<MODULE4>
<EXECUTE>YES</EXECUTE>
<FILENAME>kettenparameter</FILENAME>
<SOURCE>neuesFcap</SOURCE>
<CONNECTIONASSUMECLOSEDAFTER>0</CONNECTIONASSUMECLOSEDAFTER>
<PAYLOADENTROPY>
<NUMFIRSTBYTES>96</NUMFIRSTBYTES>
<NUMPACKETS>4</NUMPACKETS>
</PAYLOADENTROPY>
<HURST>
<INTERVALNANOSECS>100000</INTERVALNANOSECS>
<MININTERVALS>10</MININTERVALS>
<MAXSPLITS>20</MAXSPLITS>
<INCLUDEGLOBAL>YES</INCLUDEGLOBAL>
<FILENAME>hurstglobal</FILENAME>
</HURST>
</MODULE4>
</CONFIG>

```

Quelltexte

Erklärung

Ich erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.