Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Datenschutzkonforme Anonymisierung von Datenverkehr auf einem Vermittlungssystem

Studienarbeit am Institut für Telematik
Prof. Dr. M. Zitterbart
Fakultät für Informatik
Universität Karlsruhe (TH)

von

cand. inform.
Christoph Mayer

Betreuer:
Prof. Dr. M. Zitterbart
Dipl.-Inform. Thomas Gamer
Dr.-Ing. Marcus Schöller

Tag der Anmeldung: 1. März 2006
Tag der Abgabe: 15. Juli 2006

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 15. Juli 2006

Inhaltsverzeichnis

1. Einleitung

Logdateien von Angriffserkennungssystemen bzw. Aufzeichnungen von Netzwerkverkehr, sogenannte Netzwerktraces, werden in hohem Maße für Forschung und Industrie benötigt. Netzwerkverkehr, welcher auf einem Vermittlungssystem mitgeschnittener wird, ist grundlegend um Angriffe zu analysieren, nachzuspielen und unbekannte Angriffe zu erkennen. Allgemein kann mit Netzwerkverkehr das Verhalten der Netzkomponenten und von Benutzern analysiert werden. Die Entwicklung von Angriffserkennungssystemen benötigt Netzwerkverkehr für die Analyse und Evaluation des Systems. Idealerweise handelt es sich dabei um realen Verkehr, da dieser das Netz und die Benutzer des Netzes am besten widerspiegelt, mögliche unbekannte Angriffe enthält, sowie besondere Varianten von Angriffen, die in dieser Form bisher nicht beobachtet wurden.
Um mitgeschnittenen, realen Netzwerkverkehr verwenden zu können, müssen datenschutzrechtliche Grundlagen erfüllt sein. Daher müssen die Mitschnitte, unabhängig davon ob es sich um Netzwerkdumps oder Logdateien handelt, einer Anonymisierung unterzogen werden. Diese Anonymisierung muss sicher, schnell und robust sein, die wichtigsten Protokolle unterstützen und leicht auf neue Protokolle und Anonymisierungsprimitiven erweiterbar sein.

1.1 Zielsetzung der Arbeit

Ziel dieser Arbeit ist es, ein System zur Anonymisierung bzw. Transformation von Netzwerktraces zu entwickeln, mit welchem Anonymisierungen auf verschiedenen Schichten des ISO/OSI-Modells durchgeführt werden können. Es ist notwendig, dass die Transformation leicht auf neue Protokolle erweitert werden kann und neue Anonymisierungsprimitiven einfach eingebaut werden können. Protokolle müssen beliebig gekapselt werden können, daher ist eine loose Kopplung der Protokollschichten nötig. Die Verarbeitung der Netzwerkdaten muss sowohl im Online-Modus (live mitgeschnittene Daten) als auch im Offline-Modus (Anonymisierung von Daten auf einem Hintergrundspeicher) möglich sein. Hohe Verarbeitungsgeschwindigkeiten sind dabei sehr wichtig, um das System effizient unter realen Bedingungen einsetzen zu können.

1.2 Gliederung der Arbeit

Kapitel 2 führt in die Notwendigkeit von Tracedaten ein, analysiert die rechtlichen Bestimmungen zur Aufnahme und Verwendung von Netzwerkmitschnitten, gibt eine Überblick über sensitive Attribute in ausgewählten Protokollen und stellt Anonymisierungsprimitiven vor. Weiterhin werden in diesem Kapitel verwandte Arbeiten vorgestellt. In Kapitel 3 wird, ausgehend von den Analysen in Kapitel 2, der Entwurf des Anonymisierungsframework vorgestellt. Dieser Entwurf wird dann in Kapitel 4 implementiert. In Kapitel 5 wird eine Evaluierung des Frameworks durchgeführt, welche Aspekte des Betriebs betrachtet und Leistungsmessungen – auch in Bezug auf Optimierungen – darstellt. Dabei wird die Verwendung auf einem Vermittlungssystem, auf welchem hohe Datenraten zu beobachten sind, verdeutlicht. Kapitel 6 fasst die Arbeit zusammen und gibt einen Ausblick.

2. Analyse

Die Problematik der Anonymisierung von Tracedaten besteht aus mehreren Teilproblemen. Zum einen müssen die datenschutzrechtlichen Bestimmungen bezüglich Datenaufnahme und Datenveröffentlichung zwingend eingehalten werden, weiterhin müssen jedoch je nach Anwendungsfall über den Datenschutz hinausgehende Anonymisierungen angewandt werden, um sicherzustellen, dass das Netzwerk und dessen Nutzer geschützt sind. Welche Anonymisierungen konkret vorgenommen werden, hängt sehr stark von den im Datenstrom auftretenden Protokollen und dem Vertrauensverhältnis zwischen der Aufnehmenden Instanz und der Instanz, an welche die Daten weitergegeben werden, ab. Für die Anonymisierung selbst existieren verschiedene Anonymisierungsprimitiven, welche ein unterschiedliches Maß and Sicherheit bieten.
In den folgenden Abschnitten werden nach Erläuterung des Sinn und Zweck von Tracedaten in Kapitel 2.1 in Kapitel 2.2 die Bestimmungen des Datenschutzes zur Aufnahme und Verwendung von Tracedaten analysiert. Darauf folgend werden in Kapitel 2.3 für die wichtigsten Protokolle sensitive Attribute betrachtet und in Kapitel 2.4 Anonymisierungsprimitiven vorgestellt, um diese zu anonymisieren. Mögliche Angriffe auf die Anonymisierungsprimitiven werden in Kapitel 2.5 vorgestellt. Schließlich werden in Kapitel 2.6 vorhandene Arbeiten betrachtet und die Nachteile der vorhandenen Ansätze im Vergleich zu dem in dieser Arbeit entwickelten System beschrieben. Die Anforderungen an ein Anonymisierungsframework werden in Kapitel 2.7 definiert.

2.1 Notwendigkeit von Tracedaten

Die folgenden Abschnitte sollen an Hand von zwei beispielhaften Bereichen die Notwendigkeit von Netzdaten erläutern. Zum einen werden in Kapitel 2.1.1 Datenbanken mit Tracedaten und zum anderen in Kapitel 2.1.2 die Verwendung von Tracedaten in der Sicherheitsforschung betrachtet.

2.1.1 Tracedatenbanken

Tracedatenbanken bestehen aus Sammlungen verschiedener Aufnahmen von Netzdaten. Idealerweise stellt eine Tracedatenbank eine Vielzahl an unterschiedlichen

Netzdaten zur Verfügung. Je nach Tag und Uhrzeit der Aufnahme, Dauer, Geschwindigkeit des aufgenommenen Links und Diversität der Protokolle, haben die unterschiedlichen Netzdaten unterschiedlichen Nutzen für verschiedene Forschergruppen. Es gibt heute sehr wenige öffentliche Tracedatenbanken im Internet. Die beiden bekanntesten sind:

- The Internet Traffic Archive [ita]
- WIDE project [Kenj]

Das Internet Traffic Archive enthält Tracedateien, welche zwischen Februar 1996 und April 2000 aufgenommen wurden. Die Daten sind also schon recht alt und spiegeln nicht mehr die heutigen Eigenschaften von Protokollen und Nutzern wider. Die Tracedaten wurden mit verschiedenen Programmen, darunter Tcpdpriv [Mins] und Sanitize [Paxs], anonymisiert. Manche Traces wurden auch mit Programmen anonymisiert, welche nicht das Tcpdump-Format schreiben können. Sie sind daher im ASCII-Format und für die Weiterverarbeitung ungeeignet.
Die Tracedateien des WIDE projects wurden zwischen 1999 und 2006 aufgenommen. Sie wurden durch das Programm Wide-Tcpdpriv [MAWI] anonymisiert. Die Traces wurden meist über einen Zeitraum von einem Tag aufgenommen und in Dateien von ca. 200 MB aufgeteilt.
Eine Linksammlung verschiedener kleinerer Tracedatenbanken ist unter [Schu] zu finden.

2.1.2 Traces in der Sicherheitsforschung

Logdateien von Firewalls und anderen Angriffserkennungssystemen enthalten nur bekannte Angriffe, da sie meist signaturbasiert arbeiten. Tracedateien können hingegen auch unbekannte Angriffe beinhalten, da alle Pakete aufgezeichnet werden, also auch diejenigen, die einem noch unbekannten Angriff entspringen. Durch Tracedaten ist es also möglich, neue und unbekannte Angriffe zu erkennen und zu analysieren. Analysesysteme, welche die weltweite Ausbreitung von Angriffen und Würmern mit Hilfe von Logdateien erkennen, sind bereits im Einsatz [Syma]. Diese Systeme basieren auf der Hilfe von Systemadministratoren, welche ihre Logdateien von Firewalls und Angriffserkennungssystemen in anonymisierter Form zur Verfügung stellen. Durch die Analyse der Logdateien ist es dann möglich, die Ausbreitung von Angriffen und Würmern aus einer globalen Sicht zu verfolgen. Dieser Austausch an Log- und Tracedateien, welche unternehmensinterne Informationen darstellen, ist jedoch bisher nur in sehr geringem Umfang ausgeprägt. Durch die Analyse von Tracedateien können beispielsweise neue Angriffe und Würmer entdeckt werden, welche bisher nicht von Sicherheitssystemen erkannt werden und damit auch nicht in den Logdateien von Firewalls und Angriffserkennungssystemen auftauchen. Unbekannte Angriffe, sogenannte *Zero Day Exploits*, sind mehr und mehr eine starke Bedrohung. Auch der Zeitraum zwischen der Veröffentlichung einer Sicherheitslücke und deren Ausnutzung wird immer geringer, so dass eine frühe Erkennung von Angriffen notwendig wird [zero]. Der Austausch von sicherheitsrelevanten Informationen ist unter Angreifern sehr verbreitet, dadurch werden Sicherheitslücken schnell im Untergrund verbreitet, weiterentwickelt und sogar verkauft [Labo]. Durch dieses Vorgehen sind

neue und noch unveröffentlichte Angriffe oft schon sehr gut ausgearbeitet und können zu ernsten Schäden führen.
Mit Hilfe von Tracedateien könnte ein zu [Syma] ähnliches System aufgebaut werden, welches auf der Analyse von Tracedateien basiert und damit auch unbekannte Angriffe erkennen könnte. An speziellen Punkten des Internet bzw. durch die Mithilfe von freiwilligen Personen, würden Tracedateien aufgenommen werden, welche an einer zentralen Stelle gesammelt werden. Die auf diese Weise gesammelten Tracedateien müssten dann einer anomaliebasierten, über Tracedateien hinweg aggregierenden Analyse unterzogen werden, um unbekannte Angriffe, Würmer und Viren zu identifizieren. Für die Aufnahme und Weitergabe von Tracedateien ist wiederum die Anonymisierung ein notwendiger vorangehender Schritt.

2.2 Datenschutz und unternehmerische Sicht

2.2.1 Datenschutzrechtliche Bestimmungen

Eine Anonymisierung der Netzwerkpakete muss den datenschutzrechtlichen Bestimmungen des Telekommunikationsgesetzes [tele] genügen. Das Fernmeldegeheimnis §88, Abschnitt 3 schreibt vor, dass es "untersagt ist, sich oder anderen über das für die geschäftsmäßige Erbringung der Telekommunikationsdienste einschließlich des Schutzes ihrer technischen Systeme erforderliche Maß hinaus Kenntnis vom Inhalt oder den näheren Umständen der Telekommunikation zu verschaffen". Die Veröffentlichung von gesammelten Daten ist laut §96, Abschnitt 3 erlaubt, falls die Daten anonymisiert werden. Laut §100, Abschnitt 1 gilt, dass "zum Erkennen, Eingrenzen oder Beseitigen von Störungen oder Fehlern an Telekommunikationsanlagen die Bestandsdaten und Verkehrsdaten der Teilnehmer und Nutzer erhoben und verwendet werden dürfen". Daher ist eine Verwendung der Daten für die Verwendung in Angriffserkennungssystemen erlaubt. Insbesondere dürfen in Forschungseinrichtung laut Bundesdatenschutzgesetz [bund] §40, Abschnitt 1 "für Zwecke der wissenschaftlichen Forschung erhobene oder gespeicherte personenbezogene Daten [...] nur für Zwecke der wissenschaftlichen Forschung verarbeitet oder genutzt werden". Abschnitt 2 erläutert die Bedingungen für eine Nutzung: "Die personenbezogenen Daten sind zu anonymisieren, sobald dies nach dem Forschungszweck möglich ist". Auch eine Definition der Anonymisierung ist im Bundesdatenschutzgesetz §3, Abschnitt 6 enthalten: "Anonymisieren ist das Verändern personenbezogener Daten derart, dass die Einzelangaben über persönliche oder sachliche Verhältnisse nicht mehr oder nur mit einem unverhältnismäßig großen Aufwand an Zeit, Kosten und Arbeitskraft einer bestimmten oder bestimmbaren natürlichen Person zugeordnet werden können".

2.2.2 Unternehmerische Bedenken

Unternehmen sehen in einer Freigabe ihrer Tracedateien eine Gefahr für ihre Firmengeheimnisse und Daten. Diese Bedenken sind berechtigt, da ohne entsprechende Anonymisierung der Daten sensitive Informationen in den veröffentlichten Dateien erhalten bleiben. Die Freigabe von Trace- oder Logdateien wirft laut [Adam04] folgende Problematik für das Unternehmen auf:

- Angreifer können sich ein Bild der internen Netztopologie verschaffen

- Die Daten enthalten mögliche Informationen über Engpässe im Netzwerk und damit mögliche Ziele für DDoS Angriffe
- Der Datenteil der Pakete kann Benutzernamen und Passwörter im Klartext enthalten
- Es ist möglich mit Würmern oder Trojanern infizierte Clients zu identifizieren, welche dann als Angriffsziel verwendet werden können
- Der Datenteil der Pakete enthält private Informationen der Netzwerknutzer
- In den Headern höherer Protokolle sind oft Softwareversionen der Server enthalten

Diese Informationen können von Angreifern verwendet werden um das Netzwerk zu kompromittieren. Um die Gefahren für das Unternehmen zu minimieren, ist eine sichere Anonymisierung notwendig.

2.3 Sensitive Attribute in Paketen

Im Folgenden werden die Attribute verschiedener Protokolle auf die in Abschnitt 2.2.1 genannten datenschutzrechtlichen Bestimmungen und auf den Schutz von Unternehmen, seinen Nutzern, Netzwerken und Daten geprüft. Dabei werden ausgewählte Protokolle beginnend auf Schicht 2 bis hin zur Schicht 4 des ISO/OSI-Modells betrachtet. Die Reihenfolge der Attribute entspricht, soweit hier unterschieden werden kann, ihrer Sensitivität.

2.3.1 Ethernet-Paket

Die meist verbreitete Netzwerktechnologie stellt heute das Ethernet dar. Daher wird eine Link Layer Encapsulation nach IEEE 802.3 [8023] auf Schicht 2 des ISO/OSI-Modells betrachtet. Folgende Felder eines Ethernet-Pakets sind zu diskutieren:

- Hardwareadresse der Netzwerkkarte der Quelle (6 Byte)
- Hardwareadresse der Netzwerkkarte des Ziels (6 Byte)

Die Hardwareadresse (MAC-Adresse) ist eine weltweit eindeutige feste Adresse, welche der Hersteller der Netzwerkkarte vergibt. Die ersten drei Byte beschreiben den Hersteller der Karte, die letzten drei Byte identifizieren die Karte eindeutig.
Ein Angreifer kann die Information aus der MAC-Adresse beispielsweise verwenden, um herauszufinden, welche Hardware im Netzwerk verwendet wird. Zum Beispiel deutet eine MAC-Adresse mit den ersten drei Byte `00-01-63` auf Hardwarekomponenten der Firma Cisco Systems, Inc. [cisc] hin. Dadurch ist es dem Angreifer möglich, herauszufinden welche Sicherheitslücke auf diesem System möglicherweise existiert. Allein in der Securityfocus Datenbank [Secu] befinden sich über 150 Sicherheitslücken von Systemen der Firma Cisco. Es bietet sich also mit dieser Information ein weites Spektrum an Möglichkeiten um entsprechende Systeme anzugreifen.
Weiterhin identifiziert die MAC-Adresse den physikalischen Teilnehmer am Netzwerk global eindeutig und muss somit laut Bundesdatenschutzgesetz (siehe Abschnitt 2.2.1) anonymisiert werden.

2.3.2 IP-Paket

Auf Schicht 3 des ISO/OSI-Modells ist das Internet-Protokoll IPv4 heute weit verbreitet und wird in den meisten Internet- und Netzwerkanwendungen eingesetzt. Daher wird ein IP-Datagramm nach [Post81b] betrachtet. Folgende Felder sind als sensitiv einzustufen:

- Adresse, welche den Sender des Pakets bestimmt (4 Byte)
- Adresse, welche den Empfänger des Pakets bestimmt (4 Byte)
- Optionen
- Identifikationsnummer für das Paket (2 Byte)
- Time-to-Live Zähler, welcher bei jedem Hop dekrementiert wird (1 Byte)
- Prüfsumme über den Header (2 Byte)

Die Sender- und Empfänger-IP-Adressen sind hoch sensibel, da sie die Nutzer im Netzwerk identifizieren. Sie müssen unbedingt anonymisiert werden, um die Privatsphäre der Nutzer und des Netzwerks zu sichern.
Die Optionen eines IP-Pakets sind sehr vielfältig und können in bestimmten Situationen – zum Beispiel bei Verwendung der Record Route Option – wieder IP-Adressen beinhalten. Somit muss sichergestellt werden, dass diese IP-Adressen nicht in unanonymisierter Form beibehalten sondern entsprechend anonymisiert werden. Werden die IP-Optionen später nicht benötigt, so können diese auch komplett aus dem Paket gelöscht werden.
Die Identifikationsnummer kann von einem Angreifer für eine Known Plaintext Attacke verwendet werden. Dazu sendet er ein Paket mit einer eindeutig von ihm bestimmten Identifikationsnummer in das Netzwerk, über welches er mehr Informationen herausfinden möchte. Die Original-IP-Adresse des Hosts ist dem Angreifer somit bekannt. Sein Ziel ist es, die dazu anonymisierte Adresse herauszufinden. In den veröffentlichten Traces sucht der Angreifer nach dem speziellen Paket. Auf Grund des gut wieder auffindbaren Merkmals der Identifikationsnummer ist dies leicht möglich. Durch mehrere solche Angriffe zu verschiedenen Zeitpunkten können Nichteindeutigkeiten schnell aufgelöst werden, falls in einem Trace die vom Angreifer vergebene Identifikationsnummer mehrfach vorkommt. Mit diesem Angriff kann der Angreifer die anonymisierte IP-Adresse zu einer realen IP-Adresse aufdecken. Nach mehreren erfolgreichen solchen Angriffen kann die Anonymisierungsprimitive für IP-Adressen selbst angegriffen werden. Es ist offensichtlich, dass Known Plaintext Attacken, wie die eben vorgestellten nur für Hosts möglich sind, die aus einem anderen Netzwerk (zum Beispiel dem Internet), in welchem sich der Angreifer befindet, erreichbar sind.
Das Time-to-live Feld (TTL) gibt Auskunft über die Struktur des Netzwerks. Jeder Router - allgemeiner jeder Hop - dekrementiert das TTL Feld. Es ist somit über die Auswertung dieses Feldes möglich eine Grobstruktur des Netzes aufzubauen, da bekannt ist, wie weit die einzelnen Hosts vom Eingang in das Netzwerk entfernt sind [Loza].
Die Prüfsumme muss bei einer Transformation neu berechnet werden. Wird sie nach der Anonymisierung beibehalten, so kann ein Angreifer durch eine Brute Force Attacke versuchen, die Originalwerte der Felder des Pakets zu berechnen. Er muss dazu

herausfinden, welche Felder seit der ursprünglichen Prüfsumme verändert wurden. Diese veränderten Felder werden nun mit zufälligen Werten besetzt und die Prüfsumme neu berechnet. Ist die berechnete Prüfsumme korrekt, so ist die Wahrscheinlichkeit sehr hoch, dass der zufällige Wert dem realen Wert des Ursprungspakets entspricht. Diese Attacke ist nur sinnvoll möglich, wenn nur eine geringe Anzahl an Feldern verändert wurde. Durch Neuberechnung der Prüfsumme nach einer Paket-Transformation kann dieser Angriff verhindert werden.

2.3.3 TCP-Paket

Folgende Felder eines nach [Post81c] definierten TCP-Pakets sind zu betrachten:

- Portnummer des Senders, von dem das Paket gesendet wurde (2 Byte)
- Portnummer, zu dem das Paket beim Empfänger gesendet wird (2 Byte)
- Sequenznummer (4 Byte)
- Acknowledge-Nummer (4 Byte)
- Prüfsumme (2 Byte)

Die Portnummer des Senders bzw. Empfängers des TCP-Pakets gibt Aufschluss über die im Netzwerk verwendeten Dienste. Die Portnummer 25 beispielsweise deutet mit hoher Wahrscheinlichkeit auf einen Email-Server hin. Dadurch kann ein Angreifer mögliche Angriffsziele im Netzwerk identifizieren.
Mittels der Sequenz- und Acknowledge-Nummer hat der Angreifer wie bei der Identifikationsnummer des IP-Paket (siehe Abschnitt 2.3.2) die Möglichkeit einer Known Plaintext Attacke und kann somit die Anonymisierung unterlaufen.
Die Prüfsumme des TCP-Headers umfasst im Gegensatz zu der des IP-Pakets auch die Daten, welche das Paket mit sich trägt. Dadurch wird eine Brute Force Attacke interessanter, da meist die Daten sensitive Informationen enthalten, jedoch auch schwieriger, weil die Vielfalt der Daten weitaus größer ist als die eines anderen Feldes. Es ist somit notwendig die Prüfsumme nach Anonymisierung des Pakets neu zu berechnen, um einer Attacke auf die Prüfsumme vorzubeugen.

2.3.4 UDP-Paket

Sensitive Felder eines UDP-Pakets nach [Post80] sind:

- Portnummer des Senders, von dem das Paket gesendet wurde (2 Byte)
- Portnummer, zu dem das Paket beim Empfänger gesendet wird (2 Byte)
- Prüfsumme (2 Byte)

Hier gelten für die Portnummern sowie für die Prüfsumme die gleichen Überlegungen wie bei einem TCP-Paket (siehe Abschnitt 2.3.3).

2.3.5 ICMP-Paket

Es wird ein ICMP-Packet nach [Post81a] betrachtet. ICMP wird benötigt, um über Probleme im Netzwerk zu informieren. Dabei gibt es viele verschiedene Nachrichten, um die große Vielfalt an möglichen Netzwerkfehlern abzudecken. Bei einigen Nachrichten wird als Datenteil des ICMP-Pakets wiederum ein komplettes IP-Paket mitgesendet. In diesem Fällen muss das eingekapselte IP-Paket wie ein normales IP-Paket behandelt werden und entsprechend anonymisiert werden.

Destination Unreachable Message

Eine Destination Unreachable Message wird verwendet, um dem Sender eines Pakets mitzuteilen, dass der Empfänger des Pakets nicht erreichbar ist. Der Datenteil des ICMP-Pakets ist in diesem Fall der IP-Header und die ersten 64 Bit des Protokolls, welches in dem IP-Paket gekapselt sind. Hier gelten die Überlegungen zum IP-Paket in Abschnitt 2.3.2 und abhängig von dem im IP-Paket gekapselten Protokoll die jeweiligen Abschnitte zu dem entsprechenden Protokoll aus Abschnitt 2.3.

Time Exceeded Message

Die Time Exceeded Message wird verwendet, wenn beispielsweise das TTL-Feld des IP-Pakets den Wert 0 erreicht hat. Das entsprechende Gateway, welches diesen Sachverhalt entdeckt hat, sendet eine Time Exceeded Message an den Sender des Pakets. Der Datenteil enthält ebenso wie bei der Destination Unreachable Message in Abschnitt 2.3.5 den IP-Header und die ersten 64 Bit der Daten des IP-Pakets. Es gelten somit dieselben Überlegungen wie bei der Destination Unreachable Message.

Parameter Problem Message

Die Problem Message wird verwendet, wenn bei einem Host Probleme mit einem Feld des Pakets auftraten. Wie bei der Destination Unreachable Message und Time Exceeded Message werden wieder der IP-Header und die ersten 64 Bit der Daten im IP-Paket mitgesendet. Es gelten somit die gleichen Überlegungen wie bei der Destination Unreachable Message.

Source Quench Message

Die Source Quench Message wird verwendet, um anzuzeigen, dass das Gateway, welches das Paket verarbeiten sollte, nicht genügend Pufferplatz zur Verfügung hatte. Als Datenteil werden der IP-Header und die ersten 64 Bits der Daten im IP-Paket mitgesendet. Es gelten die Überlegungen wie bei der Destination Unreachable Message.

Redirect Message

Die Redirect Message Nachricht wird verwendet, wenn ein Paket auf Grund eines kürzeren Weges umgeleitet wurde. Da wieder der IP-Header und die ersten 64 Bit der IP-Daten enthalten sind, gelten wie gleichen Überlegungen wie bei der Destination Unreachable Message.

Echo and Echo Reply Message

Mit dieser ICMP Nachricht wird das bekannte *Ping* Programm realisiert. Ein Host sendet eine Echo Message, der Empfänger antwortet darauf mit einer Echo Reply Message. Die Daten, die in dem Echo Paket enthalten sind, müssen den Daten in der Echo Reply Message gleichen. Um Echo-Nachrichten auf Echo-Reply-Nachrichten zuzuordnen, werden in den Paketen Sequenznummern verwendet. Diese können verwendet werden, um Pakete im anonymisierten Datenstrom aufzufinden und die Anonymisierung der IP-Adresse anzugreifen (siehe Abschnitt 2.5).

Timestamp or Timestamp Reply Message

Dieses Paar von Nachrichten wird zur Zeitmessung zwischen zwei Hosts verwendet. Die Daten enthalten daher jeweils eine Timestamp. Mit Hilfe dieser Timestamps kann ein Angreifer Informationen über die Entfernung der beiden Hosts erhalten oder auch über die Last, welche auf der Verbindung zwischen den beiden Hosts besteht. Damit kann ein Angreifer sich eine Übersicht der Subnetze schaffen und stark belastete Übertragungsabschnitte identifizieren, welche für Denial of Service Angriffe anfällig sind.

Information Request or Information Reply Message

Mit der Information Request Nachricht kann ein Host herausfinden, welche Adresse das Netzwerk besitzt, auf dem er sich befindet. Die Nachrichten können somit IP-Adressen in besitzen und müssen anonymisiert werden.

2.3.6 Unbekannte Protokolle und Payload

Unbekannte oder vom Anonymisierungsframework nicht unterstützte Protokolle müssen komplett anonymisiert werden, da nicht bekannt ist, ob sie möglicherweise sensitive Informationen enthalten.
Da die Vielfalt der Protokolle ab Schicht fünf sehr breit ist, können nicht alle Protokolle unterstützt werden. Hier muss abhängig von der späteren Verwendung der anonymisierten Daten entschieden werden, ob ein neuer Protokoll-Parser und Protokoll-Transformator eingebaut wird (siehe Abschnitt 3 und 4) oder dieser Teil des Pakets als Datenpuffer bzw. unbekannte Daten gehandhabt wird und mit Nullen oder zufälligen Daten überschrieben wird (siehe Abschnitt 2.4.1).

2.3.7 Übersicht sensitiver Felder

Tabelle 2.1 gibt nochmals eine Übersicht der sensitiven Felder in den genannten Paketen. Dabei wurde zwischen den einzelnen zu transformierenden Feldern in ihrer Wichtigkeit unterschieden in Bezug auf das Datenschutzgesetz und die Sicht des Unternehmens, in dessen Netzwerk die Daten aufgezeichnet werden.
Ein Wert von *0* kennzeichnet ein nicht sensitives Feld, ein Wert von *3* ein sehr sensitives Feld.
D steht für die Sicht des Datenschutzgesetzes und *U* steht für die Sicht des Unternehmens.

Wie aus der Auflistung in Tabelle 2.1 zu sehen ist, sind aus datenschutzrechtlicher Sicht nur drei der insgesamt zehn für sensitiv befundenen Felder zu anonymisieren.

Feld	ETH	IP	TCP	UDP	ICMP	D	U
MAC-Adresse	x					3	3
Identifikationsnummer		x				0	1
Time-to-live		x				0	2
Prüfsumme (nur Header)		x				0	1
Prüfsumme (mit Daten)			x	x	x	0	2
IP-Adresse		x				3	3
Portnummern			x	x		0	2
Sequenznummer			x			0	1
Acknowledgenummer			x			0	1
Daten			x	x	x	3	3

Tabelle 2.1: Übersicht sensitiver Attribute aus Datenschutz- und Unternehmens-Sicht

Die Sicht des Unternehmens ist sehr viel restriktiver und zielt darauf ab, so wenig Informationen wie möglich preis zu geben, unabhängig davon, ob es sich um private Daten der Nutzer oder Informationen über das Netz handelt. Das Datenschutzgesetz sieht nur den Schutz des Benutzers, das Unternehmen selbst beabsichtigt hingegen weiterhin den Schutz des Netzwerks und der damit verbundenen Unternehmensressourcen und Unternehmensgeheimnisse. Weiterhin ist zu beachten, dass das Datenschutzgesetz sich nur auf die Transformation vorhandener Daten stützt. Der Sachverhalt, dass ein Angreifer versucht, den Paketstrom zu seinen Gunsten zu manipulieren, wird nicht betrachtet. Daher muss die Identifikationsnummer aus datenschutzrechtlicher Sicht nicht anonymisiert werden.

2.4 Anonymisierung sensitiver Felder

In diesem Abschnitt werden die als sensitiv eingestuften Felder und die möglichen Transformationen betrachtet, welche die Anonymität der Daten wahren. Dabei interessiert auch das Verhältnis von Sicherheit und Nutzen. Sicherheit bedeutet das Level an Anonymität, welches die Transformation erzielt. Der Nutzen ist das Level an Information, welches in den Daten nach einer Transformation erhalten bleibt. Sicherheit verhält sich somit antiproportional zum Nutzen der Daten. Je mehr Informationen in den Paketen anonymisiert werden, umso höher ist die Sicherheit, entsprechend sinkt damit jedoch auch der Nutzen, da viele Felder und Protokolle nicht mehr für die Weiterverarbeitung bzw. Analyse verwendbar sind und auch statistische Daten verloren gehen. Die Wahl der Anonymisierung ist somit bei jeder Verwendung der Daten neu zu bedenken und entsprechend anzupassen, damit eine

höchstmögliche Sicherheit erreicht wird und der Nutzen nicht höher als notwendig ist.

2.4.1 Zufalls- und Konstantentransformation

Ein sensitiver Datenblock wird durch eine zufällige Bitfolge oder eine Folge einer Konstanten - zum Beispiel Nullen - überschrieben. Dadurch geht die komplette Information - bis auf die Länge des Blocks - verloren. Das Verfahren ist für die meisten sensitiven Felder nicht anwendbar, da die transformierten Daten keinen Nutzen mehr aufweisen.

$$f(x \in \{0,1\}) = \begin{cases} k, & Konstantentransformation, k = const \in \{0,1\} \\ a, & Randomtransformation, a = Rnd() \in \{0,1\} \end{cases} \tag{2.1}$$

Diese Transformation eignet sich vor allem für höhere Protokollschichten, welche vom Parser nicht unterstützt werden, und für reine Datenblöcke eines Protokolls. Nicht unterstützte Protokolle werden als Datenblock angesehen und mit zufälligen oder konstanten Bits überschrieben. Ein auf diese Weise transformiertes Protokoll stellt dann unter Umständen kein wohlgeformtes Paket mehr dar. Da die Datenstruktur bei dieser Transformation nicht beachtet wird, werden bestimmte Felder, (z. B. Versionsnummern) welche die Wohlgeformtheit des Pakets beeinflussen, nicht korrekt gesetzt.
Um reine Datenblöcke zu transformieren, wie zum Beispiel die Daten eines TCP-Pakets, ist dies das beste Verfahren. Falls auch die Länge des Feldes als sensitiv betrachtet wird, so muss der Datenblock anonymisiert und auf eine zufällige Länge verlängert oder reduziert werden.
Um den Speicherplatz der Pakete auf dem Hintergrundspeicher zu minimieren, ist es auch möglich, die durch die beschriebene Anonymisierung transformierten Daten zu entfernen und nur die Längenangabe des einkapselnden Pakets zu erhalten.

2.4.2 Tabellenbasierte Transformation

Durch eine Tabelle kann ein eindeutiges Mapping definiert werden. Beispielsweise kann eine IP-Adresse a.b.c.d immer auf die gleiche IP-Adresse w.x.y.z. transformiert werden. Dabei kann die Transformationsfunktion beliebig erzeugt werden und muss nicht einem bestimmten Muster folgen. Wird in diesem Fall ein bestimmtes Mapping bekannt, so bleiben die restlichen geheim, da die Transformation keinem Muster folgt.
Die tabellenbasierte Transformation wird nun formal beschrieben: sei Q der Raum der zu anonymisierenden Elemente (dies kann beispielsweise der Raum der IP-Adressen sein) und $T = Q \times Q$ die Transformationstabelle, welche zu jedem $x \in Q$ das transformierte Element $x' \in Q$ liefert. Offensichtlich besitzt T eine Größe von $|Q| * |Q|$. Ist f eine Funktion einer tabellenbasierten Transformation, so gilt, dass $\forall x \in Q : \exists_1 x' \in Q$ mit $f(x) = x'$. Die Funktion f ist wie folgt definiert:

$$f(x) = \begin{cases} Q \to Q \\ x \mapsto T[x] \end{cases} \tag{2.2}$$

Die Funktion f bzw. die Tabelle T müssen nicht zwingend surjektiv sein. Diese Eigenschaft ist aber wünschenswert, da sonst die transformierten Daten an Nutzen verlieren, da sie nicht mehr eindeutig zur Analyse verwendet werden können. Falls zum Beispiel zwei verschiedene IP-Adressen x und y auf die gleiche IP-Adresse z transformiert werden, so verfälscht dies den Nutzen der Daten.
Der Vorteil einer immer gleich bleibenden, surjektiven Transformation ist, dass die Relation zwischen zwei originalen Daten und deren transformierten Daten bestehen bleibt. Wenn beispielsweise IP-Adressen über eine Tabelle transformiert werden, so ist ein Flow auch im transformierten Trace korrekt erkennbar, da eine IP-Adresse x immer auf die gleiche IP-Adresse y transformiert wird.
Das Verfahren ist mit einem hohen Aufwand für die Erstellung der Tabellen verbunden. Weiterhin stellen die Tabellen geheime Informationen dar, da sie das Mapping definieren und somit auch zur Rückwärtstransformation f^{-1} verwendet werden können. Es ist also unbedingt darauf zu achten, dass die Tabellen gesichert werden. Um verschiedene Traces vergleichbar zu machen, müssen bei der Transformation die gleichen Tabellen verwendet werden. Sollen Traces parallel auf mehreren Rechnern transformiert werden, so müssen die Tabellen für alle Rechner zugänglich sein. Der durch dieses Verfahren resultierende Aufwand birgt Gefahren, da die Verteilung und Sicherung der Tabellen komplex werden kann.

2.4.3 Fortlaufende Nummerierung

Bei einer fortlaufenden Nummerierung werden Daten entsprechend ihrem sequentiellen Vorkommen durchnummeriert. Beispielsweise wäre dies beim FTP-Protokoll anwendbar. Der Benutzername würde dann in der Form *User1*, *User2* und das Passwort als *Passwort1*, *Passwort2* transformiert. Auch IP-Adressen können der Reihe nach transformiert werden. Ab einer definierten Startadresse wird für jede neu auftretende Adresse die Transformations-Adresse inkrementiert. Es wird also ein statisches Element s benötigt, welches das letzte Element speichert. Die fortlaufende Nummerierung wird beschrieben durch:

$$f(x) = s; s = s + 1; \tag{2.3}$$

Fortlaufende Nummerierung kann in Verbindung mit dem tabellenbasierten Verfahren (siehe Abschnitt 2.4.2) verwendet werden, in dem alle erstellen Nummerierungen in die Tabelle eingetragen werden. Dadurch ist ein eindeutiges Mapping möglich und es können erneut auftretende Elemente wieder mit dem gleichen anonymisierten Element ersetzt werden. Durch Verwendung des tabellenbasierten Verfahrens resultieren wieder die Probleme, welche in Abschnitt 2.4.2 besprochen wurden.

2.4.4 Verrauschen

Es ist möglich, vorhandene Traces zusätzlich zu anderen Transformationen zu verrauschen und damit die Daten weiter zu anonymisieren. Dabei werden nur bestimmte Felder verrauscht, so dass die Paketstruktur erhalten bleibt (Längen- und Versionsfelder müssen beispielsweise korrekt bleiben). Es wird ein Paket x mit einer Länge von l Bit betrachtet.

Eine Rauschfunktion f ersetzt an zufälligen Stellen i das Bit $x[i]$ durch ein zufällig berechnetes:

$$x[i] = x \in Rnd() \in \{0,1\}, i \in Rnd() \in [0,\ldots,l-1]$$

Die Stärke des Rauschen ist als Abstandsmaß der originalen zu den verrauschten Daten definierbar und errechnet sich aus den unterschiedlichen Bits. Sei x das Original-Paket und x' das verrauschte Paket. Ohne Beschränkung der Allgemeinheit besitzen beide die Länge l. Der Abstand D zwischen x und x' ist dann rekursiv definiert durch:

$$D(x,x') = \begin{cases} 0, l = 1 \wedge x[0] = x'[0] \\ 1, l = 1 \wedge x[0] \neq x'[0] \\ 0 + D(x-1, x'-1), l \neq 1 \wedge x[l-1] = x'[l-1] \\ 1 + D(x-1, x'-1), l \neq 1 \wedge x[l-1] \neq x'[l-1] \end{cases} \tag{2.4}$$

Falls zum Beispiel Adressfelder verrauscht werden, ist der zugehörige Flow des Pakets nicht mehr korrekt zu identifizieren. Es ist also darauf zu achten, nur Felder zu verrauschen, welche nicht den Nutzen der Daten verringern.

2.4.5 Hashen

Ein effizienter und einfacher Weg, die Notwendigkeit von Tabellen zu umgehen und trotzdem Eindeutigkeit und eine nicht lineare Transformation zu erhalten, sind Hashfunktionen. Eine Funktion $f : K \rightarrow S, f(x) = x'$ heist Hashfunktion, falls $|K| \geq |S|$. Eine Hashfunktion ist eindeutig, liefert also bei erneuter Anwendung immer den gleichen Zielwert x'. Zu einer Hashfunktion f gibt es keine Umkehrfunktion f^{-1}. Eine Änderung von x um ein einzelnes Bit resultiert in einer nicht vorhersehbaren Änderung des Hashwertes x'.
Attribute wie IP-Adressen und Portnummern sind beispielsweise gute Kandidaten für Transformationen mit Hashfunktionen. Eine transformierte IP-Adresse bleibt im gesamten Trace eindeutig und und Flows können weiterhin, auch über verschiedenen Traces hinweg, erkannt werden.
Da der Zahlenraum der Portnummern mit $2^{16} = 65\,536$ Elementen relativ klein ist, ist die Hashfunktion anfällig für einen Brute Force Angriff. Ob das Hashen von IP-Adressen anfällig für Brute Force Angriffe ist, hängt von der Art des Hashens ab. Das komplette Hashen der IP-Adresse ist auf Grund der Größe des IP-Adressraum mit $2^{32} = 4\,294\,967\,296$ als relativ sicher zu betrachten. Byteweises Hashen der IP-Adressen ist hingegen anfällig für Brute Force Angriffe, da sich die Anzahl der Elemente dadurch auf $2^8 = 256$ reduziert. Wird der Netzwerk- und Host-Teil der IP-Adresse einzeln gehasht, dann muss der Angreifer in Kenntnis der Anzahl an Host- und Netzwerk-Bits sein, um die Anonymisierung angreifen zu können. Je nach Anzahl der Bits im Host-Teil, liegt die Anzahl an möglichen IP-Adressen theoretisch zwischen $2^1 = 2$ und $2^{31} = 2\,147\,483\,648$. Entsprechend ist die Möglichkeit eins Brute Force Angriffs abhängig von der Anzahl an Host-Bits.

2.4.6 Schlüsselbasiertes Hashen

Um das Problem der Brute Force Angriffe - wie in Abschnitt 2.4.5 erläutert - zu umgehen, empfiehlt sich die Anwendung von verschlüsselten Hashfunktionen, im englischen *keyed-hash message authentication code* oder *HMAC* genannt [Touc97]. Hashfunktionen wie MD5 oder SHA1 werden verwendet und in Verbindung mit einem Schlüssel gehasht. Dadurch entsteht eine kryptographische Hashfunktion, welche nur mit Kenntnis des Schlüssels mit einem Brute Force Angriff angreifbar ist.
Der Schlüssel muss geheim gehalten werden, dies zeigt sich jedoch einfacher als z. B. ganze Tabellen bei einer tabellenbasierten Transformation (siehe Abschnitt 2.4.2) geheim zu halten. Eine Anonymisierungstabelle kann recht groß sein und muss somit auf einem Datenträger gespeichert werden. Ein einfacher Schlüssel hingegen muss nicht gespeichert werden, sondern kann von der verantwortlichen Person gemerkt werden.
Bei Verwendung des gleichen Schlüssels liefert ein HMAC für einen Wert immer den gleichen Hashwert. Somit ist es möglich, verschiedenen Traces mit dem gleichen Schlüssel zu verschiedenen Zeitpunkten oder auf verschiedenen Rechner zu transformieren und vergleichbare, transformierte Traces zu erhalten.

2.4.7 Präfixerhaltende Transformation

Gerade bei IP-Adressen ist es oft eine wünschenswerte Eigenschaft der Transformation, dass Hosts, die im gleichen Subnetz sind, nach der Transformation wieder im gleichen, transformierten Subnetz liegen. Um diese Eigenschaft mit einer Hashfunktion zu erreichen, gibt es zwei Möglichkeiten:

1. Die einzelnen Bytes der IP-Adresse werden einzeln gehasht (klassenbasierte Adressierung)
2. Es ist für jede IP-Adresse bekannt, welche Bits den Network- und welche Bits den Host-Teil der Adresse bestimmen. Network- und Host-Bits werden unabhängig voneinander gehasht (CIDR-basierte Adressierung)
3. Die Transformation formt die IP-Adressen auf solch eine Weise mathematisch um, dass die Transformation automatisch präfixerhaltend ist

Im ersten Fall ist es nur möglich, *Class A, B* oder *C* Netzwerke korrekt zu hashen, da die Grenze von Netzwerk- zu Host-Bits an einer Bytegrenze endet. Im zweiten Fall müssen Informationen über die Grenzen von Netzwerk- und Host-Bits zum Hashen herangezogen werden. Diese müssen gegebenenfalls aus der Routingtabelle oder einer anderen Quelle entnommen werden. Der dritte Fall benötigt am wenigsten Informationen, ist jedoch rechnerisch aufwendiger. Verfahren zur präfixerhaltenden Anonymisierung wurden unter anderem in [XFAM01] entwickelt.

2.5 Angriffe auf die Transformationsprimitiven

In diesem Abschnitt werden mögliche Angriffe auf die Transformationsprimitiven betrachtet. Dabei wird von einem Angreifer ausgegangen, welcher die veröffentlichten Tracedaten verwendet, um Informationen über das Netzwerk bzw. die Daten zu erfahren.
Einen guten Überblick möglicher Attacken gibt auch [Adam04].

2.5.1 Fingerabdrücke

Der Angriff des Fingerprinting beschreibt die Methode durch bekannte Informationen Rückschlüsse auf die Anonymisierung zu ziehen. Falls ein Angreifer zum Beispiel weiß, dass das Unternehmen nur einen aus dem Internet erreichbaren Webserver betreibt, kann er durch Analyse der Pakete mit Ziel-Port 80 herausfinden, welche anonymisierte IP-Adresse dem Server zuzuordnen ist. Da die reale IP-Adresse des Webservers bekannt ist, kann er somit ein Mapping aufdecken und schließlich versuchen die Anonymisierungsprimitive für IP-Adressen selbst anzugreifen.

2.5.2 Strukturerkennung

Wenn eine IP-Adresse herausgefunden wurde, sind bei einer präfixerhaltenden Anonymisierung von IP-Adressen die originalen k Bit anderer Adressen bekannt, wenn sich diese mit der anonymisierten IP-Adresse ein k Bit Präfix teilen.

2.5.3 Angriff durch bekannte Zuordnung

Falls mehrere Traces des gleichen Netzwerks unterschiedlich transformiert wurden, ist es möglich Zuordnungen zu treffen und so unanonymisierte, sensitive Daten anonymisierten Daten zuzuordnen. Wurden beispielsweise in Trace A Portnummern anonymisiert und in Trace B nicht, so ist es möglich durch Häufigkeitsanalyse die Portnummern der beiden Traces aufeinander abzugleichen und somit die Portnummern im anonymisierten Trace aufzudecken.

2.5.4 Daten einschleusen

Ein Angreifer kann selbst Pakete in das Netzwerk schicken. In den veröffentlichten, anonymisierten Traces versucht er seine eigenen Pakete zu finden. Dies kann durch Setzen selten genutzter Flags oder Optionen oder durch Setzen von Sequenznummern erleichtert werden. Der Angreifer kann so herausfinden, welche Pakete von der Firewall des Netzwerks gefiltert werden, und erhält wertvolle Informationen über die Sicherheitsvorkehrungen des Unternehmens.
In den veröffentlichten Traces kann der Angreifer dann die anonymisierte IP-Adresse der realen zuordnen. Da er die Pakete ja selbst geschickt hat, ist ihm die originale IP-Adresse bekannt. Somit kann die Anonymisierung Stück für Stück gebrochen werden, indem alle möglichen Hosts des Netzwerks erkannt werden.

2.5.5 Brute Force Angriffe

Falls zur Anonymisierung Hashfunktionen verwendet wurden, können diese durch Brute Force Angriffe gebrochen werden. Der Aufwand einer solchen Attacke hängt vom attackierten Attribut ab, ist jedoch für Portnummern oder IP-Adressen durchaus möglich (siehe Abschnitt 2.4.5).

2.5.6 Bekannter Klartext

Wurde ein Mapping eines Attributs erkannt, kann durch eine Known Plaintext Attacke die Anonymisierungsprimitive selbst angegriffen werden. Verläuft dies erfolgreich, können schließlich alle Mappings des Attributs aufgedeckt werden. Dies ist nicht bei allen Anonymisierungsprimitiven möglich. Beispielsweise sind schlüsselbasierte Hashfunktionen (siehe Abschnitt 2.4.6) nicht mit dieser Methode angreifbar.

2.5.7 Statistische Angriffe

Durch statistische Analyse der Größe von Paketen und Verteilung der Portnummern ist es möglich, die Anonymisierung der Portnummern anzugreifen. Verschiedene Dienste besitzen charakteristische Eigenschaften bzgl. Größe der Pakete und Auftrittshäufigkeiten. Der meist verwendete Dienst ist beispielsweise HTTP auf Port 80. Somit kann der anonymisierte Trace auf Häufigkeit der verschiedenen Portnummern analysiert werden und eine Zuordnung getroffen werden. Auch die Paketgröße gibt Aufschluss über den verwendeten Dienst. Beispielsweise sind IMAP-Pakete sehr groß wohingegen Pakete einer SSH-Sitzung eher klein sind.

2.5.8 Potentielle Angriffsziele

Unabhängig von den verwendeten Transformationsprimitiven hat ein Angreifer die Möglichkeit potentielle Angriffsziele im Netzwerk zu finden. IP-Adressen, die sehr oft angesprochen werden, deuten auf wichtige Knoten hin, beispielsweise Datenbank- oder Email-Server. Hier ist es besonders wichtig durch die Anonymisierung diese Hosts entsprechend zu schützen. Auch ist es über Analyse der Traces möglich, von Trojanern oder Würmern befallene Hosts zu erkennen, da diese meist auf sonst nicht verwendeten Ports auf Verbindungen warten und diese Verbindungen in den Traces erkannt werden können.

2.6 Existierende Lösungsansätze

In [Adam04] wurde die Notwendigkeit von öffentlich zugänglichen Log- und Tracedatenbanken erörtert. Es wurden zwei Metriken eingeführt, die Nützlichkeitsmetrik und die Sicherheitsmetrik. Vorhandene Werkzeuge zur Anonymisierung wurden vorgestellt und mögliche Angriffsformen beschrieben. Weiterhin beschreibt die Arbeit zwei Aspekte der Anonymisierung: den Schutz des Netzwerks und den Schutz der Benutzer.
Pang et al haben in [Ruom03] eine erweiterbare, programmierbare Umgebung beschrieben, mit welcher Pakete geparst werden und durch Rückruffunktionen Anonymisierungsprimitiven implementiert werden können. Die Arbeit bezieht sich mehr auf die Problematik bei höheren Protokollschichten und wie diese anonymisiert werden können.
In [XFAM01] beschreiben Xu et al Möglichkeiten zur präfixerhaltenden Anonymisierung von IP-Adressen. Sie beschreiben die Möglichkeit, mit schlüsselbasierten Hashfunktionen eine sichere, präfixerhaltende Anonymisierung durchzuführen.
Mogul zeigte in [Mogu02] die Problematik von Anonymisierung und schlechten Formaten von Logdateien auf. Das Common Log Format ist nach seiner Beschreibung zu unvollständig und es fehlen grundlegende Attribute wie Timestamps. Weiterhin müssen anonymisierte Traces verifiziert werden. Bisher existieren auch keine Verfahren, welche optimal zwischen Nutzen und Sicherheit skalieren, da sie entweder zu viel Information entfernen oder zu viel Information in den transformierten Traces belassen.
In [Kenj] wurden die Probleme einer Aufnahmen von Gigabit-Verkehr und die Probleme vorhandener Anonymisierungstools wie Tcpdpriv beschrieben.
Peuhkuri zeigte in [Peuh01] eine Methode zum Anonymisieren und Komprimieren von Traces auf. Es wird eine auf Flows basierte Komprimierung vorgestellt, welche

von der Annahme ausgeht, dass viele Pakete in einer Kommunikationssitzung gleiche Felder besitzen und dadurch eine hohe Redundanz in dem Trace entsteht. Über IP-Adresse, IP-Header, Protokollnummer und Portnummer wird ein Hashwert berechnet, welcher den Flow charakterisiert. Bei allen Paketen, welche zu dem gleichen Flow gehören, wird nur die Differenz zum vorherigen Paket gespeichert. Dadurch wird der Trace auf etwa 10% der originalen Größe komprimiert. Zur Anonymisierung der IP-Adresse wird Blowfish mit 128 Bit über MD5 gewählt.
In [Patr04] werden die Probleme von Log-Datenbanken und Möglichkeiten zur Anonymisierung besprochen. IP-Adressen werden mit SHA1 oder MD5 als HMAC verschlüsselt. Andere sensitive Felder werden nicht betrachtet. Zum Veröffentlichen der Log-Dateien wird ein *Random Alert Routing* vorgeschlagen, mit welchem die Log-Dateien zu zufälligen anderen Clients übertragen werden und von diesen zufällig weiter geroutet werden bis sie schließlich bei der Datenbank enden und in diese aufgenommen werden. Dadurch kann die Quelle der Daten nicht nachvollzogen werden und somit die Identität des Netzwerks, aus welchem die Logdateien entspringen, geschützt werden. Außerdem wurde ein Plugin für das Angriffserkennungssystem Snort [Roes] entwickelt, mit welchem sich Logdateien automatisch anonymisieren und anonym zur Datenbank übertragen lassen.
Bereits existierende Tools für die Trace-Anonymisierung sind:

- Tcpdpriv
- Sanitize
- Wide-Tcpdpriv
- Crypto-PAn

Das Programm Tcpdpriv [Mins] besitzt mit der A50 Option ein tabellenbasiertes Verfahren (siehe Abschnitt 2.4.2), um IP-Adressen zu anonymisieren. Dabei wird eine Tabelle von originalen und anonymisierten Adressen verwendet, welche während der Anonymisierung aufgebaut wird. Das Verfahren hat zum einen das Problem sehr viel Speicher zu verbrauchen und zum anderen können Traces nicht parallel oder zeitlich verschoben verarbeitet werden, da sie dadurch nicht mehr vergleichbar sind. Die Optionen von Tcpdpriv sind sehr vielfältig und daher fehleranfällig für falsche Bedienung, welche unter Umständen in der Freigabe von sensitiven Information enden.
Sanitize [Paxs] ist eine Kollektion von fünf Shellskripten. Die verschiedenen Skripte sind auf TCP, SYN/FIN Flags, UDP und eingebettete IP-Pakete spezialisiert. IP-Adressen werden mit einer fortlaufenden Nummerierung (siehe Abschnitt 2.4.3) anonymisiert. Dadurch können verschiedenen Traces nicht mehr miteinander verglichen werden. Außerdem ist die Ausgabe von Sanitize in ASCII Format und nicht als Tcpdump-Datei, was die Weiterverarbeitung unmöglich macht.
Wide-Tcpdpriv [MAWI] ist eine aus [Kenj] hervorgegangene Erweiterung von Tcpdpriv. In erster Linie wurden die vielfältigen und fehleranfälligen Optionen vereinfacht und einige Protokolle wie IPv6 erweitert.
Crypto-PAn [J. F] ist eine aus [XFAM01] hervorgegangene Implementierung einer präfixerhaltenden Anonymisierung von IP-Adressen. Die beigefügte Beispiel-Anwendung verwendet keine Tcpdump-Traces, sondern kann nur eine Liste von IP-Adressen mit Timestamps transformieren.

Jedes dieser Programmen hat Restriktionen. Diese sind je nach Programm die folgenden:

- Durch die Anonymisierung lassen sich Traces nicht mehr vergleichen
  - Tcpdpriv
  - Sanitize
- Die Anwendung bereitet Schwierigkeiten, welche dann in der Veröffentlichung privater Daten enden
  - Tcpdpriv
- Es werden nur wenige Protokolle adressiert
  - Tcpdpriv
  - Sanitize
  - Wide-Tcpdpriv
  - Crypto-PAn
- Es werden nur spezielle Attribute adressiert
  - Tcpdpriv
  - Sanitize
  - Wide-Tcpdpriv
  - Crypto-PAn
- Es werden keine Tcpdump-Traces verwendet bzw. die Ausgabe ist nicht im Tcpdump-Format, was die Weiterverarbeitung erschwert
  - Crypto-PAn
  - Sanitize
- Es können keine kompletten Traces automatisch transformiert werden
  - Crypto-PAn

2.7 Anforderungen an ein Anonymisierungsframework

Nach dem in den vorherigen Anschnitten die Protokolle auf ihre sensitiven Attribute analysiert (siehe Abschnitt 2.3) und Anonymisierungsprimitiven sowie Angriffe auf diese vorgestellt wurden (siehe Abschnitt 2.4 und 2.5), werden nun aus diesen Informationen die Eigenschaften für ein Framework für die Paket-Transformation abgeleitet:

- Einfache und unkomplizierte Erweiterung auf neue Protokolle

- Lose Kopplung der Protokollschichten für beliebige Kapselungen
- Leichte Bedienbarkeit
- Einfach zu implementierende Transformationsprimitiven für beliebige Protokolle und Attribute
- Möglichkeit jedes beliebige Attribut zu transformieren
- Verwaltung der Paket-Organisation und Transformation
- Pessimistische Transformation und defensives Parsen zur Sicherstellung, dass keine sensitiven Daten im transformierten Trace vorhanden sind
- Kryptographisch sichere Transformationsprimitiven
- Geringer Speicher- und CPU-Verbrauch
- Eingabe und Ausgabe als binäres Tcpdump-Format
- Automatische Anpassung von Feldern wie Länge und Prüfsumme nach einer Transformation - Pakete müssen wohlgeformt bleiben wie in [Ruom03] gefordert
- Die Netzwerkstruktur muss erhalten bleiben
- Einfach an den benötigten Nutzen anpassbare Sicherheit

2.8 Zusammenfassung

Bisher erarbeitete Anonymisierungen (siehe Abschnitt 2.6) beschränken sich meist auf wenige Attribute. Implementierungen sind schlecht erweiterbar und statisch. Dadurch kann nicht individuell eine dem Nutzen angepasste Sicherheit erreicht werden. Der Aufbau von öffentlichen Tracedatenbanken wird somit verhindert und weiterführende Forschung gehindert.

3. Entwurf

Im folgenden werden der Entwurf und die grundlegenden Prinzipien des Frameworks *Pktanon* vorgestellt. Das Framework Pktanon soll die in Abschnitt 2.7 dargelegten Anforderungen erfüllen, beliebige Transformationen erlauben, leicht erweiterbar sein und eine schnelle und sichere Anonymisierung der Netzdaten ermöglichen.

3.1 Definitionen

Eine *Paket-Klasse* ist eine Schablone für ein bestimmtes Protokoll und enthält den Protokoll-Header und, falls vorhanden, Protokoll-Optionen. Beim Parsen der Netzdaten werden auf Basis der Paket-Klassen Instanzen, sogenannte *Paket-Objekte*, erstellt, welche reale Netzwerkpakete repräsentieren. Die Einkapselung der Pakete aus den Netzdaten ineinander wird durch die Verkettung der Paket-Objekte zu einer *Paket-Kette* oder *Packet-Chain* gebildet. Die Verkettung ist linear und rein vorwärts gerichtet. Eine Verkettung zwischen zwei Paket-Objekten A und B symbolisiert somit die Einkapselung des Pakets B in das Paket A.
Abbildung 3.1 zeigt die beispielhafte Erstellung einer Packet-Chain aus einem TCP-Paket.

3.2 Pakete und Paket-Ketten

Pakete sind von einem bestimmten Protokoll-Typ und geben dem Nutzer eine einfache Schnittstelle zum Lesen und Schreiben der Attribute des jeweiligen Protokolls über `get-` und `set`-Methoden. Somit können auch Attribute, welche nur aus einem Bit bestehene wie beispielsweise Flags einfach gehandhabt werden. Plattformabhängige Eigenschaften wie Byteordnung werden automatisch gehandhabt, ebenso ist es Aufgabe einer Paket-Klasse selbständig Attribute wie Längen, Prüfsummen etc. anzupassen und dafür zu sorgen, dass das Paket bei jeder Änderungen der Attribute und Transformationen wohlgeformt bleibt. Neue Protokolle können mit wenig Aufwand hinzugefügt werden. Das Handling der Packet-Chains sorgt dafür, dass Protokolle unabhängig voneinander sind und automatisch verarbeitet werden. Dadurch ist eine beliebige Verschachtelung der Pakete möglich.
Die Packet-Chains werden automatische beim Einlesen der Netzdaten erstellt. Sie

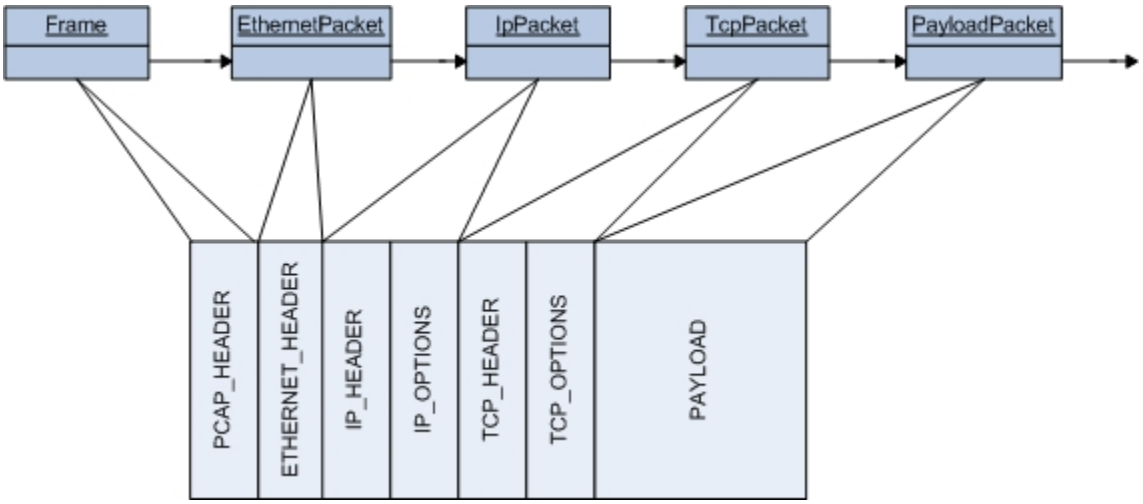

Abbildung 3.1: Paket und zugehörige Packetchain

basieren nicht zwingend auf einer Quelle wie einer Tcpdump-Datei, sondern können auch manuell erstellt werden.

3.3 Paket-Objekte

Eine Paket-Klasse muss die folgenden zwei Funktionen implementieren:

- `parsePacket`
- `assemblePacket`

Die *parsePacket*-Funktion wird vom Framework aufgerufen, um dem Paket-Objekt mitzuteilen, dass sich Daten in seinem Puffer befinden, welche in das Paket-Objekt aufzunehmen sind. Die Verwaltung der Puffer wird vom Framework selbst gehandhabt. Eine Paket-Klasse muss die Daten aus dem Puffer holen, diese parsen und die eigentlichen Informationen des Protokolls über `get`- und `set`-Methoden zur Manipulation bereitstellen. Weiterhin muss die Paket-Klasse während der Abarbeitung dieser Funktion die Attribute *nextProtocol*, *layersize* und *minProtocolSize* setzen. Damit weiß das Framework, welches das nächste innere Protokoll ist (*nextProtocol*), wie groß der Platz des Paketobjekts im Puffer ist (*layersize*) und wie groß der minimale Protokollheader der Paketobjekts ohne Optionen mindestens ist (*minProtocolSize*). Diese Attribute sind notwendig, da nur das Paket-Objekt selbst weiß, welches das nächste innere Protokoll ist und auch nur das Paket-Objekt seine eigene minimale und aktuelle Größe kennt. Beispielsweise ist bei einer Einkapselung eines TCP-Pakets in ein IP-Paket nur dem IP-Paket bekannt, dass es sich bei seinem Datenteil um ein TCP-Paket handelt. Wo das TCP-Paket beginnt, kann nur die IP-Paket-Klasse bestimmen, welche das IP-Paket parst. Daher muss die Klasse, welche das IP-Paket parst, die Informationen zum eingekapselten Protokoll und seiner eigenen Größe setzen.
Der Aufruf der Funktion *assemblePacket* verlangt vom Paket-Objekt sich selbst zusammenzubauen. Da sich die Größe des Pakets durch Manipulation verändern kann, zum Beispiel falls die Protokoll-Optionen gelöscht werden oder sich die Größe des

eingekapselten Protokolls ändert, teilt das Paketobjekt dem Framework seine neue Größe über das *layersize*-Attribut mit und kopiert dann seine Attribute - der Definition des Protokolls entsprechend - in den vom Framework bereitgestellten Puffer. Dabei muss das Paket selbst für die Wohlgeformtheit der Daten sorgen und bestimmte Attribute wie Längen und Prüfsummen anpassen.

3.4 Paket-Transformation

Die Paket-Transformation basiert auf der Idee, nicht Attribute aus einem Paket oder Puffer zu ändern oder löschen, sondern ein neues Paket desselben Typs zu erstellen und dessen Attribute zu setzen. Durch dieses *defensive Transformieren* wird sichergestellt, dass keine sensitiven Informationen in den transformierten Daten enthalten sind. Es werden also nicht direkt in den Trace-Dateien Informationen geändert, sondern diese werden von Paket-Objekten in neue Paket-Objekte übertragen.
Das Framework baut aus einer vorhandenen Paket-Kette eine neue auf, welche initial leere Pakete enthält. Transformationsfunktionen werden für jede Protokoll-Klasse definiert und dann für jedes Glied der Paket-Kette mit dem originalen und dem neuen, leeren Paket-Objekt aufgerufen. Aufgabe der Transformationsfunktion ist es nun, die gewünschten Attribute aus dem originalen Paket in das neue zu übertragen. Dabei stehen Transformationsprimitiven (siehe Abschnitt 2.4) zur Verfügung, um die Daten zu manipulieren. Da die Paket-Klassen einfache `get-` und `set`-Funktionen unterstützen, kann ein Paket-Objekt mit wenigen Zeilen Quellcode transformiert werden. Die Transformationsfunktionen sind nicht Teil der Paket-Klassen, sondern werden gesondert realisiert, um die Darstellung der Pakete über die Paket-Klassen von ihrer Transformation unabhängig zu machen.

3.5 Paket-Dump

Eine Paket-Kette kann zu jedem Zeitpunkt über das Framework in einen Puffer *gedumpt* werden. Dabei werden automatisch, rekursiv mit dem letzten Glied beginnend die einzelnen Paket-Objekte zusammengebaut. Es ist notwendig mit dem letzten Glied zu beginnen, da Paket-Objekte weiter vorne in der Kette bestimmte Längenattribute – zum Beispiel bei Änderung des Paketinhalts auf einer höheren Schicht – oder Prüfsummen anpassen müssen. Ein einziger Aufruf lässt alle Pakete sich rekursiv zusammenbauen, das Puffer-Handling wird selbständig angepasst und schließlich kann ein Puffer, welcher alle Pakete in zusammengebauter Form enthält, dem Tcpdump-Format entsprechend erweitert und in eine Datei geschrieben werden.

3.6 Zusammenfassung

Der Entwurf des Pktanon Frameworks basiert auf folgenden Grundüberlegungen:

- Einfach für neue Protokolle erweiterbar (siehe auch Abschnitt 5.2)
- Beliebige Paket-Verschachtelungen
- Sicherheit bei der Anonymisierung durch defensive Transformation

- Transformationsfunktionen sind einfach zu erstellen und einzubauen
- Verwaltung durch Paket-Ketten
- Kein direktes Manipulieren von Tracedateien

Durch diese Eigenschaften können die in Abschnitt 2.7 geforderten Punkte gewährleistet werden. Wie diese Eigenschaften genau realisiert sind, wird in Abschnitt 4 aufgezeigt.

4. Implementierung

Das Pktanon Framework wurde komplett in C++ geschrieben. Während der Entwicklung wurde darauf geachtet Pktanon unter Windows und Linux lauffähig zu halten. Pktanon wurde getestet mit

- Visual Studio 7 C++ Compiler, Windows XP
- g++ 3.3.3, Cygwin, Windows XP
- g++ 3.3.5, GNU/Linux 2.6.16

Im Folgenden werden die internen Arbeitsweisen von Pktanon sowie beispielhaft eine Transformationsfunktion vorgestellt.

4.1 Übersicht

Abbildung 4.1 zeigt den Aufbau von Pktanon (Hilfskonstrukte wurden zugunsten der Übersichtlichkeit weggelassen). Wie man sieht, besteht es nur aus wenigen Klassen. Wie sich zeigen wird, ist es dadurch jedoch sehr einfach zu handhaben und erfüllt die in 2.7 geforderten Anforderungen.

4.2 Frame-Klasse

Ein *Frame* ist die grundlegende Hülle einer Paket-Kette (siehe Abschnitt 3.2). Es enthält alle Informationen, welche auch im Recordheader `PCAP_REC_HEADER` (siehe Abschnitt 4.3) der Tcpdump-Datei zu finden sind und dient als Schnittstelle zum Lesen und Schreiben der PcapFile-Klasse (siehe Abschnitt 4.3). Ein *Frame* enthält außerdem einen Zeiger auf das erste Paket der Paket-Kette.
Die *Frame*-Klasse wird nur zum Schreiben und Lesen in bzw. aus der Datenquelle, zum Beispiel eine Tcpdump-Datei, verwendet. Alle anderen Funktionalitäten - wie Transformationen - basieren direkt auf Paket-Ketten.

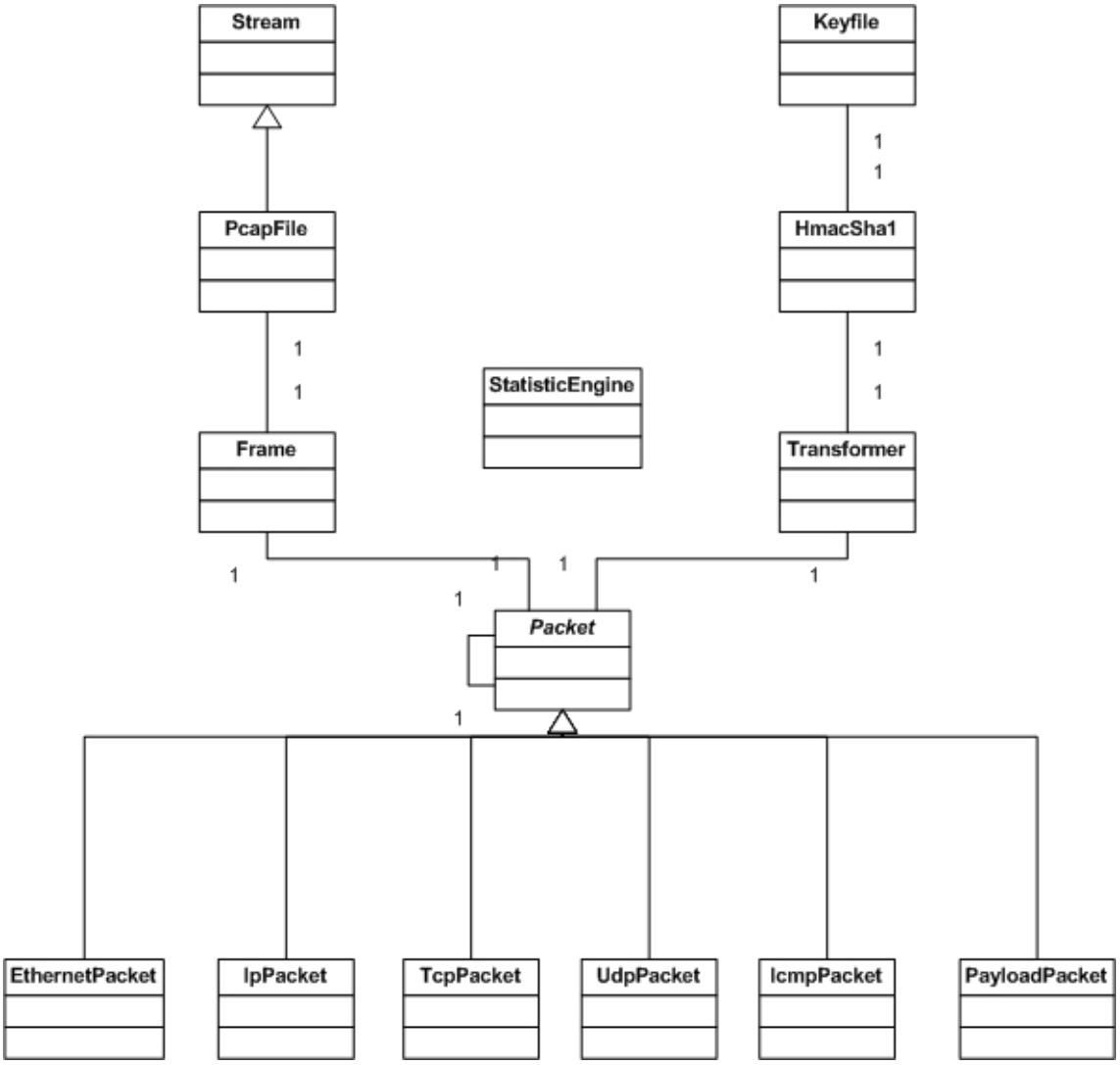

Abbildung 4.1: Aufbau des Pktanon Frameworks

Um die komplette Paket-Kette zusammenzubauen kann die Funktion

```
Packet::PAYLOAD_BUFFER Frame::getPayloadBuffer ()
```

verwendet werden. Sie fordert die einzelnen Glieder der Paket-Kette (mit dem letzten Glied beginnend) auf, ihre Daten zusammenzubauen und gibt den kompletten Puffer mit Daten an den Aufrufer zurück. Der Zusammenbau der Puffer beginnend mit dem letzten Glied der Kette ist notwendig, um geänderte Attribute wie Längen und Prüfsummen der einkapselnden Pakete berechnen zu können. Beispielsweise muss das IP-Protokoll wissen, wie groß der Inhalt seines Datenteils ist. Da sich dieser ändern kann, muss diese rückwärts gerichtete Vorgehensweise verwendet werden.

4.3 PcapFile-Klasse

Um das Format einer Tcpdump-Datei zu kapseln, wurde die *PcapFile*-Klasse implementiert. Sie basiert auf der *Stream*-Klasse, welche das Lesen und Schreiben - wahlweise aus einer Datei oder der Standardein- und Ausgabe - ermöglicht. Die Klasse *PcapFile* kann zum Lesen oder Schreiben von Trace-Dateien verwendet werden. Über die Funktionen

```
Frame* readFrame  ();
bool   writeFrame (Frame& pkt);
```

steht dafür ein einfaches Interface zur Verfügung.

Eine Tcpdump-Datei ist wie folgt aufgebaut: zu Beginn der Datei steht ein *Pcapheader*, welcher die Datei spezifiziert. Dieser ist wie folgt aufgebaut:

```
typedef struct _PCAP_FILE_HEADER {
     unsigned long   magic;          // immer 0xa1b2c3d4
     unsigned short  version_major;  // derzeit 2
     unsigned short  version_minor;  // derzeit 4
     unsigned long   thiszone;       // GMT zu lokaler Zeitkorrelation
     unsigned long   sigfigs;        // Genauigkeit der Zeitstempel
     unsigned long   snaplen;        // Maximale Länge der Pakete
     unsigned long   network;        // Typ des Netzwerks
} PCAP_FILE_HEADER,  *PPCAP_FILE_HEADER;
```

Interessant ist hier vor allem das Attribut *network*: Es steht für den Typ des Netzwerks, in welchem der Trace aufgenommen wurde. Für Ethernet ist der Wert 1 definiert. Nach dem Header folgen bis zum Ende der Datei die Pakete in roher Form. Dabei geht jedem Paket ein *Recordheader* voran, welcher wie folgt aufgebaut ist:

```
typedef struct _PCAP_REC_HEADER {
     Frame::TIME_VAL ts;             // Zeitstempel
     unsigned long   incl_len;       // Länge des Pakets in der Datei
     unsigned long   orig_len;       // Ursprüngliche Länge des Pakets
} PCAP_REC_HEADER, *PPCAP_REC_HEADER;
```

`Frame::TIME_VAL` definiert einen Zeitstempel und ist wie folgt aufgebaut:

```
typedef struct _TIME_VAL {
     long            tv_sec;         // Zeit in Sekunden
     long            tv_usec;        // Zeit in Mikrosekunden
} TIME_VAL, *PTIME_VAL;
```

Dem `PCAP_REC_HEADER` folgen die Daten des Pakets. Dabei hat das Paket in der Trace Datei die Länge `PCAP_REC_HEADER.incl_len`. Ist in der Trace Datei nicht das komplette Paket aufgenommen, so definiert `PCAP_REC_HEADER.orig_len` die Länge des Pakets, welche es auf dem Netzwerk hatte. Bei Aufnahmen sollte sichergestellt werden, dass immer das komplette Paket in den Trace aufgenommen wird - bei Tcpdump kann dies mit der Option `-s 0` sichergestellt werden, da es sonst beim Parsen der Daten unter Umständen zu Problemen kommen kann, da Teile einer Protokollschicht fehlen.

4.4 Packet-Klasse

Jede Protokoll-Klasse erbt von der virtuellen Klasse *Packet*. Diese verwaltet das komplette Puffer-Handling sowie die Erstellung der Packet-Chain. Die Puffer werden automatisch mit den für das Protokoll relevanten Daten gefüllt und stehen dann zum Parsen bereit. Ebenso wird der Puffer für das Zusammenbauen des Pakets bereitgestellt und verwaltet. Weiterhin stellt die *Packet*-Klasse eine Prüfsummen-Methode bereit, welche beispielsweise von IP, TCP, UDP und ICMP verwendet wird [1].
Die *Paket*-Klasse organisiert das Erstellen der Paket-Kette. Da jede für ein bestimmtes Protokoll implementierte Paket-Klasse von dieser Klasse abgeleitet ist, wird die Erstellung der Paket-Kette kaskadiert: jedes Paket ruft nach seiner Erstellung Basisfunktionen der *Paket*-Klasse auf, welche dafür sorgt, dass das nächste Paket erstellt wird und die Paket-Kette korrekt aufgebaut wird.

4.5 Protokoll-spezifische Klassen

Jede Protokoll-spezifische Klasse erbt von der *Packet*-Klasse. Dabei muss sie zwei virtuellen Funktionen der *Packet*-Klasse implementieren:

```
bool Packet::parsePacket    ();
void Packet::assemblePacket ();
```

Die Funktion `parsePacket` wird vom Framework aufgerufen, wenn der Puffer des Paket-Objekts gefüllt ist und zum Parsen bereit ist. Die Aufgabe des Paket-Objekts ist es dann, sich alle Informationen aus dem Puffer zu holen und in eigenen Attributen zu setzen.

[1] Die meisten dieser Protokolle verwenden nicht einfach die Prüfsumme über ihren Header und Daten. Meist wird ein Pseudo-Header erstellt und über diesen und ggf. die Daten die Prüfsumme berechnet. Zur letztendlichen Prüfsummenberechnung wird jedoch immer die genannte Funktion verwendet.

Weiterhin müssen drei Informationen in der Basis-Klasse gesetzt werden:

- Die minimale Größe, welche das Protokoll im Puffer verbraucht
- Der Typ des eingekapselten, nächsten Protokolls
- Die Länge, welche die eigene Protokollschicht im Puffer einnimmt (nach dem Parsen)

Diese Informationen besitzt nur das Paket-Objekt selbst und muss diese daher setzen. Ist das nächste Protokoll nicht bekannt, so kann dieses als `Packet::PROTO_NONE` oder `Packet::PROTO_DATA_PAYLOAD` angegeben werden.
Bei Angabe von `Packet::PROTO_NONE` wird das Parsen abgebrochen und die Paket-Kette nicht mehr länger aufgebaut. Somit muss `Packet::PROTO_NONE` auch verwendet werden, falls kein weiteres Paket mehr eingekapselt ist. Die Angabe von `Packet::PROTO_DATA_PAYLOAD` sorgt dafür, dass der restliche Teil der Daten im Puffer als reiner Datenteil angesehen werden. Es wird dann ein `PayloadPacket`-Objekt erstellt, welches die restlichen Daten beinhaltet und als Endstück an die Paket-Kette angehängt wird.
Eine Protokoll-spezifische Klasse muss `get`- und `set`-Methoden zum Lesen und Schreiben aller Attribute bereitstellen. Dies erleichtert die Handhabung der Attribute und der Transformation erheblich und sorgt für mehr Sicherheit, da die Attribute korrekt gehandhabt werden. Beispielsweise ist die Protokoll-spezifische Klasse dafür verantwortlich für Attribute, welche nur bestimmte Bits in einem Protokoll belegen, einfache Methoden für Lese- (`get`) und Schreib-Operationen (`set`) bereitzustellen. Dadurch werden Schwierigkeiten und komplizierte Handhabung von Attributen in den Transformationsprimitiven verhindert.
Die Funktion `assemblePacket` wird vom Framework aufgerufen, um dem Paket-Objekt mitzuteilen, dass die vorhandenen Daten zu einem Paket zusammengebaut werden sollen. Es ist dann Aufgabe der Paket-Klasse die Protokoll-spezifischen Attribute wie Versionsnummer, Längenangaben und Prüfsummen korrekt zu setzen und das Paket in den vom Framework bereitgestellten Puffer zu kopieren.

4.6 Transformer-Klasse

Auf Grund der einfachen `get`- und `set`-Funktionen der Paket-Klassen ist die Transformation entsprechend einfach. Für jedes Protokoll kann eine eigenen Transformationsfunktion definiert werden. Diese Funktion wird vom Framework aufgerufen, sobald ein Paket-Objekt vom entsprechenden Protokoll transformiert wird.
Die komplette Paket-Kette wird dem Transformer-Objekt übergeben. Für jedes Glied wird ein neues, leeres Paket-Objekt vom gleichen Typ erstellt. Die für dieses Protokoll definierte Transformationsfunktion wird mit dem originalen und dem leeren, neuen Paket-Objekt aufgerufen. Die Aufgabe der Transformationsfunktion ist nun die Attribute entsprechend im neuen Paket zu setzen.

Es wird ein Ausschnitt einer beispielhaften Transformationsfunktion für IP-Pakete betrachtet:

```
void Transformer::transformPacket (IpPacket& inpkt, IpPacket& ret)
{
     [...]
     ret.setProtocol  (inpkt.getProtocol()          );
     ret.setSourceip  (hashIp (inpkt.getSourceip()) );
     ret.setDestip    (hashIp (inpkt.getDestip  ()) );
     ret.setOptions   (NULL, 0                      );
     [...]
}
```

Zuerst wird das *Protokoll*-Attribut der vom IP-Paket gekapselten Daten übernommen. Als nächstes wird die IP-Adresse über eine HMCA-SHA1 Hashfunktion (siehe Abschnitt 4.7) anonymisiert und im neuen Paket-Objekt gesetzt. Die Optionen werden im neuen Paket nicht gesetzt, da sie als sensitiv gelten (siehe Abschnitt 2.3.2). Über einfache Transformationsprimitiven wie `hashIp ()` wird die Transformation vereinfacht und neue Transformationsprimitiven können einfach in die Transformer-Klasse eingebaut werden.

4.7 HmacSha1- und Keyfile-Klasse

Für die Anonymisierung von IP- und MAC-Adressen wird ein HMAC-SHA1 [Reid] verwendet. Um den geheimen Schlüssel zu setzen, kann eine Datei, welche den Schlüssel enthält, als Kommandozeile angegeben werden. Diese wird von der *Keyfile*-Klasse eingelesen und dem HMAC-SHA1 Algorithmus übergeben. Die Schlüssellänge beträgt 512 Bit. Bei Angabe eines längeren Schlüssels wird dieser mit SHA1 gehasht und es wird der daraus resultierende 160 Bit lange Schlüssel verwendet.
Wie bereits vorweggenommen wurde, beträgt die Länge des resultierenden Hashwerts des HMAC-SHA1 160 Bit. Sind die zu transformierenden Daten länger als 160 Bit, so wird der Hashwert im Puffer aneinander gereiht bis er die Länge der Daten besitzt. Sind die zu transformierenden Daten nur k Bit lang und k ist kürzer als 160 Bit, so werden nur die ersten k Bit des 160 Bit Hashwert verwendet. Dieses Abschneiden des Hashwertes beeinträchtigt die Sicherheit nicht (siehe [FIPS02] Seite 73).

5. Evaluierung

5.1 Betrieb

Pktanon muss mit mindestens drei Kommandozeilenparametern aufgerufen werden. Diese notwendigen Parameter sind:

- Quelle der Daten: Dies kann eine Tcpdump Trace-Datei sein oder das Schlüsselwort `stdin`. Bei Angabe von `stdin` liest Pktanon von der Standardeingabe
- Ziel der transformierten Daten: Dies kann ein Dateiname sein oder das Schlüsselwort `stdout`. Bei Angabe eines Dateinamens wird eine neue Datei angelegt und in diese werden die anonymisierten Daten im Tcpdump-Format geschrieben. Existiert die Datei bereits, so wird sie überschrieben. Bei Angabe von `stdout` wird in die Standardausgabe geschrieben
- Quelle des Schlüssels für HMAC-SHA1: Dies ist eine Datei, welche den Schlüssel für den HMAC-SHA1 enthält. Der Schlüssel besitzt eine maximale Länge von 512 Bit. Falls der Schlüssel länger als 512 Bit ist, wird er über SHA1 gehasht und der resultierende, 160 Bit lange Hashwert wird als Schlüssel verwendet (siehe Abschnitt 4.7)

Weiterhin gibt es vier optionale Parameter um das Verhalten von Pktanon zu beeinflussen:

- `byteip`: Hashe die IP-Adressen byteweise und cache die gehashten Bytes
- `cidrip`: Hash die IP-Adressen nach Netzwerk- und Host-Bits einzeln und speichere die gehashten Netzwerk-Bits. Bei Verwendung dieses Hashverfahrens muss eine Datei *routes.txt* im gleichen Verzeichnis wie Pktanon zu finden sein welche die CIDR-Informationen enthält
- `bytemac`: Hashe die MAC-Adressen byteweise und cache die gehashten Bytes

- `measure`: Jede Sekunde wird eine Messmarke erstellt. Die Messmarken werden nach Beendigung des Programms ausgegeben[1]

Es ist zu beachten, dass sich die Optionen `byteip` und `cidrip` gegenseitig ausschließen und nur eine der beiden verwendet werden kann.

Eine Messmarke beinhaltet die folgenden Informationen:

1. Anzahl an bisher verarbeiteten Byte (Pcapheader werden nicht gerechnet)
2. Anzahl verarbeiteter Frames
3. Verstrichene Dauer in Sekunden
4. Derzeitige Geschwindigkeit in Byte pro Sekunde
5. Derzeitige Geschwindigkeit in Pakete pro Sekunde
6. Derzeitige Dauer in Sekunden die für die Verarbeitung eines Pakets benötigt wird

5.1.1 Verwendung von Pktanon mit Tcpdump

Falls Netzwerkverkehr mit Tcpdump oder Windump aufgenommen wird, empfiehlt es sich die Ausgabe der Daten von Tcpdump direkt in Pktanon weiterzuleiten. Damit werden nur anonymisierte Daten auf die Festplatte geschrieben, was die datenschutzrechtlichen Bedenken minimiert.
Mit folgendem Befehl startet Tcpdump und schreibt die rohen Binärdaten in die Standardausgabe:

```
tcpdump -w -
```

Folgender Befehl startet Pktanon, welches von der Standardausgabe liest und in eine Datei schreibt:

```
pktanon stdin ausgabedatei.trace schluessel.key
```

Die gesamte Kommandozeile, um die Ausgabe von Tcpdump nach Pktanon zu weiterzuleiten, lautet somit

```
tcpdump -w - | pktanon stdin ausgabedatei.trace schluessel.key
```

Es empfiehlt sich TCPdump mit den Optionen `-U` und `-s 0` zu betreiben. Die Option `-U` bewirkt, dass sobald ein Paket komplett in Tcpdump angekommen ist, dieses sofort in die Standardausgabe geschrieben wird. Falls diese Option nicht angegeben ist, wartet Tcpdump bis der interne Puffer gefüllt ist bevor die Daten in die Standardausgabe geschrieben werden. Die Option `-s 0` teilt Tcpdump mit, jedes Paket komplett zu speichern und nicht, wie üblich, nur die ersten 68 Byte. Auf diese Weise kann der komplette Paketstrom aufgenommen werden.

[1] Da die Anzahl der Messmarken sehr groß werden kann, empfiehlt es sich die Konsolenausgabe von Pktanon mit dem Operator > in eine Datei weiterzuleiten.

5.1.2 Anonymisierungshost

Mit der Möglichkeit die Ausgabe von Pktanon in die Standardausgabe zu schreiben, ist es in Verbindung mit Tcpdump und Tcpreplay möglich, einen *Anonymisierungshost* zu bauen. Ein solcher Host anonymisiert eingehende Daten *on the fly* und sendet die anonymisierten Daten weiter beispielsweise zu einem Analysehost, welcher den Datenverkehr auf Anomalien und Angriffe untersucht. Abbildung 5.1 zeigt das Prinzip eines Anonymisierungshosts. Auf der Netzwerkkarte empfangene Daten werden mit Tcpdump aufgenommen und direkt zu Pktanon weitergeleitet, welches eine Anonymisierung vornimmt. Die Ausgabe von Pktanon wird als Eingabe von Tcpreplay verwendet, welches die anonymisierten Pakete wieder auf das Netzwerk schreibt.
Die komplette Kommandozeile zum Betrieb des Anonymisierungshosts lautet:

```
tcpdump -U -s 0 -i eth0 -w -      | \
 pktanon stdin stdout keyfile.key | \
 tcpreplay -i eth1
```

Dabei ist `eth0` das Interface auf dem die Daten ankommen und `eth1` das Interface, auf welches die anonymisierten Daten geschrieben werden sollen.
Beispielsweise kann ein solcher Aufbau verwendet werden, um ein reales Netzwerk zu analysieren, ohne dabei reale Netzdaten preis zu geben. Die Netzdaten werden anonymisiert zum Analysehost weitergeleitet, welcher den Datenverkehr analysieren kann. Der Betreiber des Betriebsnetzwerks kann dabei einer anderen Firma bzw. Instanz die Analyse überlassen ohne dabei sensitive Informationen, welche sich in den Netzdaten befinden, preisgeben zu müssen.

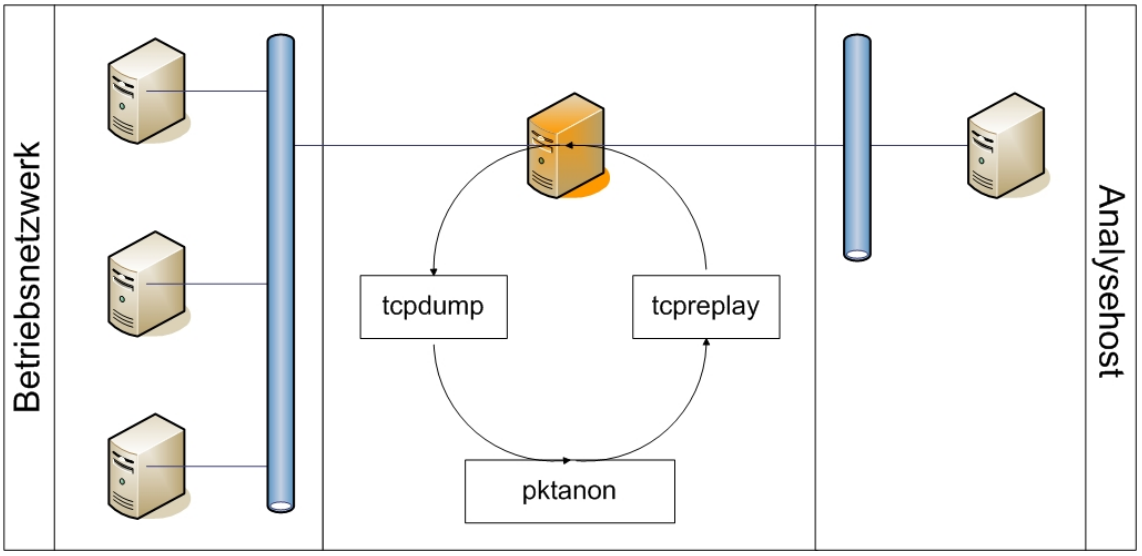

Abbildung 5.1: Aufbau eines Anonymisierungshost

5.2 Erweiterbarkeit

In Abschnitt 2.7 wurde die leichte Erweiterbarkeit von Pktanon gefordert. Es wurde bereits in Abschnitt 4.5 gezeigt, dass nur wenige Funktionen implementiert werden müssen, um ein neues Protokoll zu unterstützen. Diese Funktionen sind `parsePacket`

und `assemblepacket`.
Hier soll nun gezeigt werden, dass es ohne großen Aufwand möglich ist, spezielle Pakete wie in ICMP gekapselte IP-Pakete und IP-in-IP [Simp95] [Perk96] zu unterstützen (siehe auch Abschnitt 2.3.5).

5.2.1 IP-Pakete in ICMP

Das ICMP-Protokoll kapselt sehr häufig IP-Pakete in seinem Payload, wie in Abschnitt 2.3.5 beschrieben. Um diese IP-Pakete nicht als Datenteil zu anonymisieren und damit wertvolle Informationen über Fehler im Netzwerk zu verlieren, soll das Pktanon Framework eingekapselte IP-Pakete in ICMP-Nachrichten wie ganz normale IP-Pakete behandeln und entsprechend anonymisieren. Dazu muss der Quellcode um einige Zeilen erweitert werden. Der bisherige Quellcode der `parsePacket` Methode der `ICMP`-Klasse sieht wie folgt aus:

```
bool IcmpPacket::parsePacket()
{
     memcpy (&header, buffer, sizeof (ICMP_HEADER));

     header.checksum = swap16 (header.checksum);
     layersize       = sizeof (ICMP_HEADER);
     nextProtocol    = Packet::PROTO_DATA_PAYLOAD;

     return true;
}
```

Wie zu sehen ist wird das nächste Protokoll, also der Inhalt des ICMP-Pakets, als Datenteil (`Packet::PROTO_DATA_PAYLOAD`) dargestellt. Um dies zu ändern, muss der Programmcode wie folgt geändert werden:

```
bool IcmpPacket::parsePacket()
{
     memcpy (&header, buffer, sizeof (ICMP_HEADER));
     header.checksum   = swap16 (header.checksum);

     layersize = sizeof (ICMP_HEADER);

     //
     // depending on the type and code
     // we may have an IP-Packet in the data
     //

     switch (header.type) {

          case 3:  // destination unreachable message
          case 4:  // source quench message
          case 5:  // redirect message
          case 11: // time exceeded message
```

```
                case 12: // parameter problem message
                {
                     nextProtocol = Packet::PROTO_IP;
                     break;
                }
                case 0:  // echo message
                case 8:  // echo reply message
                case 13: // timestamp
                case 14: // timestamp reply
                {
                     nextProtocol = Packet::PROTO_DATA_PAYLOAD;
                     break;
                }
                case 15: // information request
                case 16: // information reply
                default:
                {
                     nextProtocol = Packet::PROTO_NONE;
                     break;
                }

          } // switch (header.type)

     return true;
}
```

Wie zu sehen ist, wird abhängig von der Nachricht des ICMP-Pakets der Typ des nächsten Pakets ermittelt. Das Framework übernimmt die weitere Arbeit, um das Paket richtig in der Packet-Chain anzuhängen und die Puffer zu verwalten.
Dies ist die einzige Änderung, die implementiert werden muss, um IP-Pakete in ICMP-Nachrichten korrekt zu unterstützen. Interne Handhabungen wie Prüfsummenberechnung etc. funktionieren durch Verwaltung des Frameworks weiterhin. Wie man sieht, sind die Änderungen minimal.

5.2.2 IP-in-IP

Anwendungen verwenden IP-in-IP Einkapselung [Simp95] [Perk96], um Pakete über einen definierten Host *X* zu *tunneln*. Host *X* entfernt dann das äußere IP-Paket, und sendet das innere, eingekapselte weiter. Diese Anwendung wird auch manchmal als *Mobiles-IP-Routing* beschrieben. Ein IP-in-IP Paket ist ein normales IP-Paket, welches als Datenteil wieder ein komplettes IP-Paket besitzt.
Um Pktanon IP-in-IP fähig zu machen, muss der Quellcode minimal erweitert werden. Es wird ein Ausschnitt der `parsePacket`-Methode der *IpPacket*-Klasse betrachtet:

```
     bool IpPacket::parsePacket()
     {
          [...]
```

```
        switch (header.protocol) {
             case IPTYPE_TCP:  nextProtocol = Packet::PROTO_TCP;
                               break;
             case IPTYPE_UDP:  nextProtocol = Packet::PROTO_UDP;
                               break;
             case IPTYPE_ICMP: nextProtocol = Packet::PROTO_ICMP;
                               break;
             default:          nextProtocol = Packet::PROTO_DATA_PAYLOAD;
                               break;
        }

        [...]
    }
```

In diesem Teil des Quellcodes wird das `protocol`-Feld des IP-Headers ausgewertet und dem Framework mitgeteilt, welches das nächste, innere Protokoll ist. Um IP-in-IP zu unterstützen, muss der Quellcode wie folgt erweitert werden:

```
    bool IpPacket::parsePacket()
    {
        [...]

        switch (header.protocol) {
             case IPTYPE_TCP:   nextProtocol = Packet::PROTO_TCP;
                                break;
             case IPTYPE_UDP:   nextProtocol = Packet::PROTO_UDP;
                                break;
             case IPTYPE_ICMP:  nextProtocol = Packet::PROTO_ICMP;
                                break;
             case IPTYPE_IP:    nextProtocol = Packet::PROTO_IP;
                                break;
             default:           nextProtocol = Packet::PROTO_DATA_PAYLOAD;
                                break;
        }

        [...]
    }
```

Es ist nur eine einzige Zeile Quellcode notwendig um die Erweiterung für IP-in-IP zu implementieren. Durch die lose Kopplung, die beliebige Verkettung von Paketen zu einer Packet-Chain und der Verwaltung durch das Framework ist es möglich, beliebige Einkapselungen von Protokollen mit geringst möglichem Aufwand zu implementieren.

5.3 Transformierte Pakete

Es werden zwei Pakete und deren dazu anonymisierte Pakete in den Tabellen 5.1 und 5.2 betrachtet. Für die Ausgabe wurde jede Protokoll-spezifische Packet-Klasse

mit einer `toString` Funktion versehen, um den Inhalt der Pakete einfach lesbar und damit besser vergleichbar zu machen. Die beiden Pakete stammen aus einer Anmeldung eines FTP-Client an einem FTP-Server.
Die Werte auf der linken Seite der Gegenüberstellung der Tabellen 5.1 und 5.2 stellen die originalen, die auf der rechten Seite die transformierten Pakete dar. Transformierte Werte sind unterstrichen um sie besser hervorzuheben.
Folgende Transformationen wurden vorgenommen:

- HMAC-SHA1 Hash der IP- und MAC-Adressen (kompletter Hash
- Unbekannte Protokollschichten wurden als Daten behandelt
- Nulltransformation von Daten (TCP-Payload, UDP-Payload, ICMP-Payload)
- Nulltransformation von IP-Optionen

Wie zu sehen ist, sind die transformierten IP- und MAC-Adressen durch die Anwendung des HMAC-SHA1 eindeutig. Somit ist im transformierten Trace der Flow immer noch eindeutig identifizierbar. Die Daten der TCP-Pakete wurden durch Nullen ersetzt, die Länge jedoch erhalten. Prüfsummen wurden neu berechnet um Wohlgeformtheit zu garantieren.
Dies ist nur eine beispielhafte Transformation. Pktanon stellt einen Pool an Anonymisierungsprimitiven zur Verfügung, aus denen gewählt werden kann.

5.4 Optimierungen

Es werden die Funktionen im Pktanon-Framework betrachtet und versucht durch Optimierungen die Geschwindigkeit eines Trace-Run zu verbessern.
Mit einem Profiler wurden die Funktionen, welche die meiste Laufzeit verbrauchen, herausgefunden. Der größte Anteil der Laufzeit wird für das Speichermanagement (nur unter Windows), die Berechnungen der HMAC-SHA1 Funktionen und die Prüfsummenberechnung verwendet.

5.4.1 Optimierung der Speichermanagement

Für jede Protokollschicht müssen Puffer angelegt werden um die zu parsenden Daten abzulegen. Diese Aufrufe von `malloc` und `free` tragen einen großen Teil zur Laufzeit bei. Die Größen dieser Speicherblöcke beschränken sich auf wenige konstante Werte abhängig von den Protokollen. Vor allem unter Windows beeinträchtigt das ständige Anfordern und Freigeben von Speicherblöcken die Laufzeit erheblich. Unter Linux sind die zusätzlichen Laufzeiten für das Speichermanagement nicht so hoch. Offensichtlich arbeitet das Heapmanagement unter Windows nicht so effizient, wenn oft Blöcke konstanter Größe angefordert werden. Um diese Probleme zu umgehen, wurde die Speicherbibliothek *dlmalloc* [Lea] verwendet. Diese ist darauf ausgerichtet, die Blöcke effizient zu verwalten und für Blockgrößen, welche oft verwendet werden, Blöcke nicht freizugeben, sondern diese zu speichern und bei Anforderung wieder an die Anwendung zurückzugeben. Unter Windows lässt sich damit die Laufzeit erheblich reduzieren. Dass die Einbindung von dlmalloc unter Linux die Ergebnisse kaum verändert, liegt daran, dass das Heapmanagement von Linux auf dlmalloc bzw.

Original Paket	Transformiertes Paket
Ethernet packet	Ethernet packet
source mac: 00-04-0e-bd-d9-e8	source mac: 6e-99-ba-c6-9c-3e
dest mac: 00-50-fc-23-58-71	dest mac: 37-51-b5-76-47-d8
type: 0x0800	type: 0x0800
IP packet	IP packet
version: 4	version: 4
length: 20 byte	length: 20 byte
tos: 0x10	tos: 0x10
total length: 62 byte	total length: 62 bytes
ident: 0xadf6	ident: 0xadf6
flags: 0x02	flags: 0x02
fragoff: 0x0000	fragoff: 0x0000
ttl: 56	ttl: 56
protocol: 0x06	protocol: 0x06
checksum: 0x340b	checksum: 0xa749
source ip: 193.196.41.38	source ip: 149.27.42.136
dest ip: 192.168.181.21	dest ip: 251.43.50.155
options len: 0	options len: 0
options:	options:
TCP packet	TCP packet
source port: 52131	source port: 52131
dest port: 21	dest port: 21
sequencenum: 1220206961	sequencenum: 1220206961
acknum: 519753993	acknum: 519753993
headerlen: 32 byte	headerlen: 32 byte
flags: 0x0018	flags: 0x0018
windowsize: 1460	windowsize: 1460
checksum: 0xa882	checksum: 0x5315
urgent pnt: 0	urgent pnt: 0
options len: 12	options len: 12
options: 01 01 08 0a 05 9f 52 93 00 01 f2 5b	options: 01 01 08 0a 05 9f 52 93 00 01 f2 5b
Payload packet	Payload packet
length: 10	length: 10
data: 55 53 45 52 20 42 6f 62 0d 0a	data: 00 00 00 00 00 00 00 00 00 00
string: "USER Bob"	string: " "

Tabelle 5.1: Originales und transformiertes FTP-User-Paket

Original Paket		Transformiertes Paket	
Ethernet packet		Ethernet packet	
source mac:	00-04-0e-bd-d9-e8	source mac:	6e-99-ba-c6-9c-3e
dest mac:	00-50-fc-23-58-71	dest mac:	37-51-b5-76-47-d8
type:	0x0800	type:	0x0800
IP packet		IP packet	
version:	4	version:	4
length:	20 byte	length:	20 byte
tos:	0x10	tos:	0x10
total length:	75 byte	total length:	75 byte
ident:	0xadf9	ident:	0xadf9
flags:	0x02	flags:	0x02
fragoff:	0x0000	fragoff:	0x0000
ttl:	56	ttl:	56
protocol:	0x06	protocol:	0x06
checksum:	0x33fb	checksum:	0xa739
source ip:	193.196.41.38	source ip:	149.27.42.136
dest ip:	192.168.181.21	dest ip:	251.43.50.155
options len:	0	options len:	0
options:		options:	
TCP packet		TCP packe	
source port:	52131	source port:	52131
dest port:	21	dest port:	21
sequencenum:	1220206971	sequencenum:	1220206971
acknum:	519754027	acknum:	519754027
headerlen:	32 byte	headerlen:	32 byte
flags:	0x0018	flags:	0x0018
windowsize:	1460	windowsize:	1460
checksum:	0x9926	checksum:	0x487d
urgent pnt:	0	urgent pnt:	0
options len:	12	options len:	12
options:	01 01 08 0a 05 9f 5c b3 00 01 f2 99	options:	01 01 08 0a 05 9f 5c b3 00 01 f2 99
Payload packet		Payload packet	
length:	23	length:	23
data:	50 41 53 53 20 6d 79 73 65 63 72 65 74 70 61 73 73 ...	data:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...
string:	"PASS mysecretpassword"	string:	" "

Tabelle 5.2: Originales und transformiertes FTP-Password-Paket

ptmalloc beruht [Lea] [Glog] und damit bereits für diese Anforderungen die bestmögliche Leistung erreicht. Da dlmalloc nicht für den Einsatz mit mehreren Threads entwickelt wurde, sollte stattdessen ptmalloc [Glog] verwendet werden, welches auf dlmalloc basiert und die Verarbeitung mit mehreren Threads ermöglicht. Leider ist ptmalloc im Gegensatz zu seiner Basis dlmalloc nicht mehr direkt auf Windows lauffähig. Hier müssten Änderungen am Quellcode vorgenommen werden um ptmalloc unter Windows lauffähig zu machen.

5.4.2 Optimierung des HMAC-SHA1

Um die Berechnung der HMAC-SHA1 Hashes zu beschleunigen, wurden zusätzlich zum kompletten Hashen der IP- und MAC-Adressen zwei weitere Optionen implementiert, welche diese Hashfunktionen beschleunigen. Einmal ein byteweises Hashen der IP- und MAC-Adressen und ein auf der CIDR-Adressierung basiertes Hashen der IP-Adressen.

5.4.2.1 Byteweises Hashen

Die einfachste Möglichkeit eine teilweise präfixerhaltende IP-Transformationen durchzuführen, ist die Bytes der IP-Adresse einzeln zu hashen. Damit wird sichergestellt, dass Präfixe byteweise erhalten bleiben. Bei dieser Art der Transformation werden insgesamt nur 256 verschiedene Byte gehasht. Es ist somit möglich, die Hashwerte der bereits gehashten Bytes zu speichern. Zusätzlich kann man die MAC-Adressen byteweise hashen. Die IP- und MAC-Adressen verwenden dann die gleichen 256 Byte. Somit können die Hashwerte dieser 256 Byte effizient für IP- und MAC-Adressen verwaltet werden. Um eine effiziente Datenstruktur zur Verwaltung der Byte und deren Hashwerte zur Verfügung zu haben, wurde ein einfaches Byte-Array für die Hashwerte der Byte verwendet. Der Index des Arrays ist durch den Originalwert des Byte, welches zu hashen ist, bestimmt. Um anzuzeigen welche Byte bereits gehasht wurden und deren Hashwert somit bereits in dem Array vorhanden sind, werden zusätzlich Bitmasken verwendet. Das Verfahren ähnelt somit einem tabellenbasierten Verfahren (siehe Abschnitt 2.4.2). Bei dem beschriebenen Hashverfahren werden jedoch nicht alle Einträge vorberechnet, sondern nur die benötigten während der Abarbeitung berechnet und zwischengespeichert. Bei einem tabellenbasierten Verfahren werden normalerweise keine Hashfunktionen verwendet. Ein tabellenbasiertes Verfahren basiert auf der Tatsache, dass die Mappings zwischen unanonymisierten und anonymisierten Werten rein in der Tabelle vorhanden ist und sich nicht berechnen lassen. Hashverfahren verwenden Tabellen lediglich zur Zwischenspeicherung und nicht als eigentliche Quelle der Mappings.
Das byteweise Hashen kann mit der Pktanon Option `byteip` für IP-Adressen bzw. `bytemac` für MAC-Adressen eingeschalten werden.

5.4.2.2 CIDR-basiertes Hashen

Um Präfixe der IP-Adressen auf Netzwerk-Ebene zu erhalten wurde ein Hashverfahren implementiert, welches die Netzwerk-Bits von IP-Adressen beibehält. Somit sind Hosts, welche im gleichen Subnetz liegen, nach der IP-Transformation wieder im gleichen Subnetz. Das Verfahren ist nicht komplett präfixerhaltend, wie beispielsweise das in [J. F] entwickelte Verfahren, sondern behält explizit nur die Netzwerk-Bits bei. Bei einem komplett präfixerhaltenden Verfahren gleichen sich von zwei anonymisierten IP-Adressen die ersten k Bit, falls sich die unanonymisierten IP-Adressen

ein k Bit Präfix teilten.
Die Subnetzverteilung und Adressen müssen in einer Textdatei mit Namen *routes.txt* abgelegt werden und im gleichen Verzeichnis wie Pktanon zu finden sein. Das Format der Datei ist sehr einfach: pro Zeile wird eine IP-Maske mit einer CIDR basierten Adressierung gespeichert (beispielsweise 192.168.0.0/16). Diese IP-Masken werden zu Beginn eines Transformationslaufs eingelesen und in einem Binärbaum gespeichert. Für jede IP-Adresse, welche transformiert werden soll, wird in diesem Baum das längste Netzwerk-Präfix gesucht und entsprechend werden Netzwerk- und Host-Bits einzeln über die HMAC-SHA1-Funktion gehasht. Bereits gehashte Netzwerk-Bits werden im Binärbaum gespeichert und müssen somit nicht mehr gehasht werden, falls das gleiche Subnetz erneut in einer IP-Adresse auftritt. Da über die HMAC-SHA1-Funktion nur Bytes gehasht werden können, werden die Netzwerk- und Host-Bits auf die nächst höhere Anzahl an Bits auf volle Bytes erweitert. Anschließend werden diese gehasht und später wieder die Füll-Bits abgeschnitten. Bei dem CIDR-basierten Hashen handelt es sich zum einen um eine Optimierung, vielmehr jedoch ist es hiermit möglich die Subnetzverteilung der IP-Adressen beizubehalten.
Das CIDR-basierte Hashen der IP-Adressen kann mit der Option `cidrip` eingeschaltet werden. Wie beschrieben, muss sich die Datei *routes.txt* im gleichen Verzeichnis wie Pktanon befinden.

5.4.3 Optimierung der Prüfsummenberechnung

Viele Protokolle wie IP, TCP, UDP und ICMP verwenden die gleiche Prüfsummenberechnung. Dabei wird teilweise ein Pseudoheader erstellt und über diesen die Prüfsumme berechnet. Die Funktion zur Prüfsummenberechnung ist jedoch immer identisch. Die Prüfsumme ist vereinfacht eine Aufsummierung der Byte, über welche die Prüfsumme berechnet wird. Leider ist es nicht möglich diese Aufsummierung zu optimieren, da es sich um eine rein mathematische Operation handelt, welche auf das Wesentliche reduziert ist. Es wäre möglich, die Prüfsumme nicht zu berechnen und zum Beispiel auf 0 zu setzen. Dies würde die Geschwindigkeit vor allem bei großen Paketen erhöhen. Programme wie tcpreplay können bei Verwendung der Daten die Prüfsumme neu berechnen. Jedoch würde das Nichtberechnen der Prüfsumme in Pktanon den in 2.7 geforderten Bedingungen widersprechen.

5.4.4 Optimierung der Datenweitergabe von Tcpdump nach Pktanon

Bei der Live-Anonymisierung in Verbindung mit Tcpdump werden die Daten von Tcpdump nach Pktanon weitergeleitet. Der Speicher für das Zwischenlagern der Daten im System ist dabei sehr knapp und systemabhängig. Allgemein ist jedoch nie mehr Platz als für eine handvoll Pakete. Solange die Daten aus der Standardeingabe nicht abgeholt wurden, blockiert Tcpdump beim Schreiben von Daten in die Standardausgabe. Dies bedeutet, dass alle Pakete, welche in dieser Zeit auf dem Netzwerk eintreffen, von Tcpdump verworfen werden. Dadurch werden sehr oft einige Pakete verworfen, vor allem wenn der Netzverkehr in Burst eintrifft, jedoch auch bei niedrigem Datenverkehr, wenn mehrere Pakete sehr schnell hintereinander eintreffen. Um dieses Problem der vielen Paketverluste zu vermeiden, wurde der Zwischenspeicher *Stdiobuffer* programmiert, welcher zwischen Tcpdump und Pktanon gesetzt wird.

Die Größe des Zwischenspeicher kann als Parameter übergeben werden und sollte im Bereich von einigen MB liegen.

Der komplette Aufruf von Tcpdump und Pktanon in Verbindung mit Stdiobuffer sieht dann wie folgt aus:

```
tcpdump -w - | stdiobuffer 10 |  \
     pktanon stdin ausgabedatei.trace schluessel.key
```

Stdiobuffer baut hier einen Zwischenspeicher von zehn Megabyte auf um die Pakete zwischenzuspeichern. Wie sich in Abschnitt 5.5.2 zeigt, lassen sich mit dieser Methode Paketverluste reduzieren. Vor allem wenn der bisherige Paketverlust in der Größenordnung bis zu 10% liegt, kann mit der Verwendung von Stdiobuffer der Verlust in Richtung 0% reduziert werden. Bei höheren Verlustraten ergibt sich keine Verbesserung (siehe Abschnitt 5.5.2).

5.5 Messungen

Im Folgenden werden Leistungsmessungen des Pktanon Frameworks betrachtet, mit dem Ziel, Randbedingungen ausfindig zu machen, welche für den Betrieb notwendig sind.
Die Messungen wurden auf Linux 2.6.13, Pentium 4, 2.8 GHz und 2 GB RAM durchgeführt.
Eine Messung umfasst das Einlesen, Transformieren und Schreiben der Daten, wobei immer ein Paket gelesen, transformiert und danach geschrieben wird. Danach folgt das nächste Paket. Die in 5.3 verwendeten Transformationen kommen auch hier zum Einsatz.
Zur Messung werden die folgenden beiden Tracedateien verwendet:

- 200008161359.dump
- trace_1M_packets.dump

Die Datei 200008161359.dump[2] aus [Kenj] ist 140 MB (147 144 406 Byte) groß, enthält 2 095 989 IP-Pakete und 31 nicht-IP-Pakete. Nach Abzug des Pcap-Dateiheader (24 Byte) (siehe Abschnitt 4.3) und der Pcap-Recordheader (16 Byte) (siehe Abschnitt 4.3) berechnet sich die durchschnittliche Paketgröße zu

$$\frac{147\,144\,406 - ((2\,095\,989 + 31) * 16 + 24)}{2\,095\,989 + 31} \approx 52 \text{ Byte}$$

Der Trace wurde gewählt, weil er realen Internetverkehr repräsentiert und viele verschiedene Protokolle beinhaltet. Ein anderer wichtige Punkte ist, dass die Datei sehr viele kleine Pakete enthält. Dadurch kann die Performance von Pktanon mit vielen kleinen Paketen gemessen werden. Die geringe Paketgröße resultiert aus der Tatsache, dass TCPdump standardmäßig Pakete auf 68 Byte abschneidet (siehe Abschnitt 5.1.1).
Die Datei trace_1M_packets.dump ist ein selbst aufgenommener Mitschnitt einer

[2]Die Datei ist unter folgender URL verfügbar: http://tracer.csl.sony.co.jp/mawi/samplepoint-A/2000/200008161359.html

Sitzung zwischen drei Hosts in einem 100 MBit Netzwerk, in welcher Dateien über `scp` kopiert wurden. Sie ist 1 GB (1 076 586 351 Byte) groß und besitzt eine Million Pakete. Die durchschnittliche Paketgröße errechnet sich daher zu

$$\frac{1\,076\,586\,351 - (1\,000\,000 * 16 + 24)}{1\,000\,000} \approx 1\,060 \text{ Byte}$$

Sie enthält also im Gegensatz zu der Datei 200008161359.dump große Pakete und wurde gewählt, um zu testen, wie sich die Paketgröße auf die Performance auswirkt. Zur Zeitmessung wurde eine kleine statistische Engine implementiert, welche jede Sekunde eine Messmarke erstellt (siehe Abschnitt 5.1). Um die Verarbeitungszeit von Paketen zu messen, beginnt die Messung vor dem Einlesen des ersten Pakets und endet nach dem Schreiben des letzten Pakets.
Allgemein werden vier verschiedene Aufnahmemethoden betrachtet:

- Daten in eine Datei schreiben und normales Hashen von IP- und MAC-Adressen
- Daten in eine Datei schreiben und byteweises Hashen von IP- und MAC-Adressen (siehe Abschnitt 5.4.2.1)
- Daten nach `/dev/null` schreiben und normales Hashen von IP- und MAC-Adressen
- Daten nach `/dev/null` schreiben und byteweises Hashen von IP- und MAC-Adressen (siehe Abschnitt 5.4.2.1)

Diese vier verschiedenen Arten der Verarbeitung und des Schreibens werden mit zwei Arten für die Quelldaten verwendet. Einmal wird eine Datei als Quelle verwendet (siehe Abschnitt 5.5.1) (bei dieser Offline-Anonymisierung kommen die oben genannten Tracedateien zum Einsatz) und einmal TCPdump als Quelle (also eine Online-Anonymisierung. Siehe auch Abschnitt 5.5.2) verwendet. Hierbei wird die Ausgabe von TCPdump direkt über die Standardein- und Standardausgabe mit Hilfe von Stdiobuffer nach Pktanon weitergeleitet (siehe Abschnitt 5.1.1 und 5.4.4).
Für jede hier genannte Messung wurden mindestens drei Messläufe durchgeführt. Messläufe, welche Ausreissern beinhalteten, wurden nicht verwendet. Für Messgrafiken wurde jeweils der Messlauf verwendet, welcher den Durchschnitt am besten darstellt. Bei Angabe von Messzeiten in Text oder Tabellen wurden Durchschnittswerte aus allen Messläufen verwendet.

5.5.1 Messungen mit Dateiquelle

Die Tracedateien 200008161359.dump und trace_1M_packets.dump wurden, wie in Abschnitt 5.5 vorgestellt, auf vier verschiedene Arten transformiert. Abbildung 5.2 zeigt eine Übersicht der Verarbeitungsgeschwindigkeit der verschiedenen Messungen. Tabelle 5.3 gibt einen Überblick der wichtigsten Messdaten.

Die Optimierung durch byteweises Hashen und Zwischenspeichern der Ergebnisse (siehe Abschnitt 5.4.2) erhöht die Verarbeitungsgeschwindigkeit. Bei Verwendung von byteweisem Hashen zeigen sich laut Tabelle 5.3 bei der Datei trace_1M_packets.dump eine Erhöhung der Durchschnittsgeschwindigkeit um 237% bei Schreiben nach `/dev/null`

Datei	Ziel	Byteweise Hashen	Dauer [s]	Geschwindigkeit [KByte/s]			Dauer pro Paket [s]			Pakete pro Sekunde		
				min	max	∅	min	max	∅	min	max	∅
A	/dev/null	-	128	166, 25	885, 00	866, 07	0, 000060	0, 000316	0, 000063	3165	16562	16374
		+	21	1727, 47	5520, 45	5271, 04	0, 000009	0, 000031	0, 000011	32688	104121	99620
	Datei	-	130	121, 01	877, 98	855, 97	0, 000061	0, 000435	0, 000065	2299	16478	16183
		+	22	1392, 32	5312, 21	5068, 82	0, 000010	0, 000038	0, 000011	26360	100737	95805
B	/dev/null	-	70	6895, 11	18058, 59	14804, 30	0, 000069	0, 000179	0, 000071	5584	14570	14285
		+	21	22683, 79	63727, 83	50024, 22	0, 000019	0, 000055	0, 000022	18104	53840	48210
	Datei	-	98	1997, 52	14512, 30	10758, 98	0, 000083	0, 000601	0, 000119	1664	12040	10388
		+	52	5731, 92	31724, 32	20215, 72	0, 000033	0, 000171	0, 000066	5851	30718	19494

Tabelle 5.3: Übersicht der wichtigsten Messdaten (A = 200008161359.dump, B = trace_1M_packets.dump)

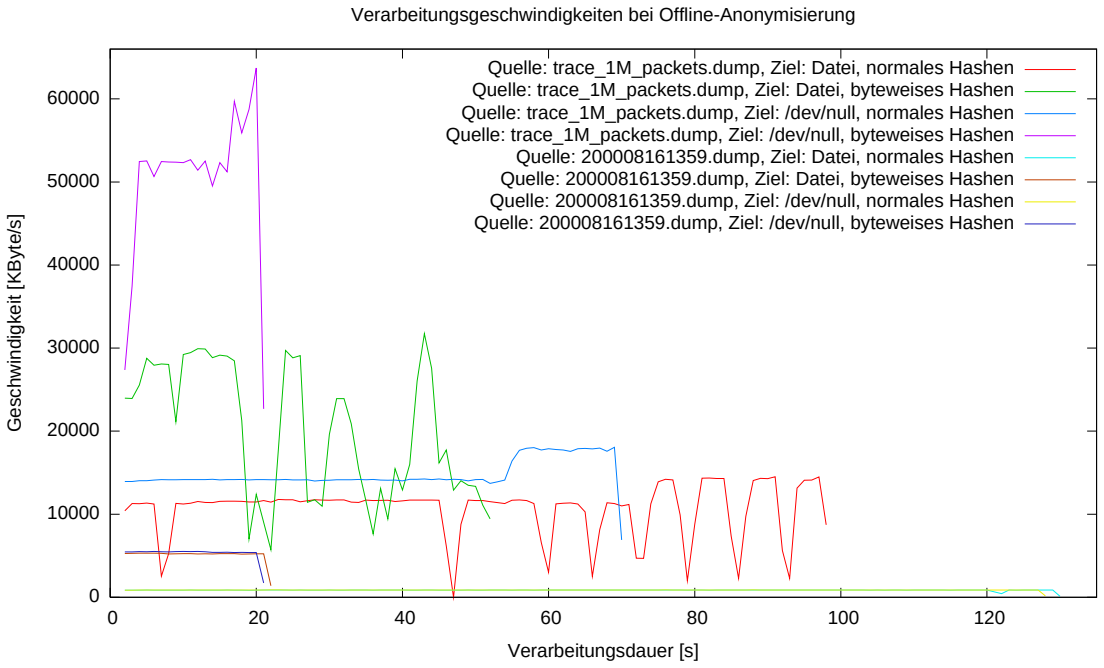

Abbildung 5.2: Verarbeitungsgeschwindigkeiten verschiedener Messungen mit Dateiquelle

und um 87% bei Schreiben in eine Datei. Die Datei 200008161359.dump erreicht eine Erhöhung der durchschnittlichen Verarbeitungsgeschwindigkeit von 508% bei Schreiben nach `/dev/null` und 492% bei Schreiben in eine Datei. Die große Erhöhung bei Verarbeitung der Datei 200008161359.dump erklärt sich durch die größere Anzahl an Paketen und der damit verbundenen höheren Anzahl an Aufrufen der Hashfunktion, im Gegensatz zu der Anzahl an Paketen in trace_1M_packets.dump. Abbildung 5.3 und 5.4 zeigen die Messungen nach Datenziel gruppiert, um die Option des byteweisen Hashens besser darzustellen.

Es wird das auf CIDR-basierte Hashverfahren (siehe Abschnitt 5.4.2.2) untersucht. Um eine möglichst direkte Gegenüberstellung der verschiedenen Hashverfahren zu erreichen, wird die Verarbeitungsdauer der Transformationen mit den verschiedenen Hashverfahren betrachtet. Tabelle 5.4 zeigt die Gegenüberstellung der Laufzeiten verschiedener Hashverfahren. Als Dateiquelle wurden dabei die ersten 100 MB der Datei trace_1M_packets.dump verwendet. Die Datei enthält genau zwei IP-Adressen welche für das CIDR-basierte Hashverfahren einmal einem /20 Netzwerk und einmal einem /10 Netzwerk zugeordnet wurden um zu prüfen, ob sich die Länge des Hostpostfix auf die Performance des CIDR-basierten Hashens auswirkt. Die Messungen wurden als einzige auf einem anderen Rechnersystem ausgeführt. Daher sind die Messzahlen dieser Messung nur in Relation zueinander vergleichbar und nicht mit den anderen Messungen.
Wie sich in Tabelle 5.4 zeigt, ist die Verarbeitungsgeschwindigkeit des CIDR-basierten Hashens der IP-Adressen unabhängig von dem zugeordneten Subnetz genauso schnell wie das komplette Hashen der IP-Adressen. Es ergibt sich also keine Geschwindigkeitserhöhung. Dies liegt zum einen daran, dass auf die nächsthöhere Anzahl an Bytes gerundet werden muss und diese gehasht werden müssten, zum anderen ent-

Hashverfahren	komplett IP	bytewise IP	cidr IP 10	cidr IP 20
komplett MAC	6299.56	8399.42	6299.56	6299.56
bytewise MAC	8399.42	14399.00	8399.42	8399.42

Tabelle 5.4: Übersicht durchschnittlicher Verarbeitungsgeschwindigkeiten (in KByte/s) unterschiedlicher Hashverfahren

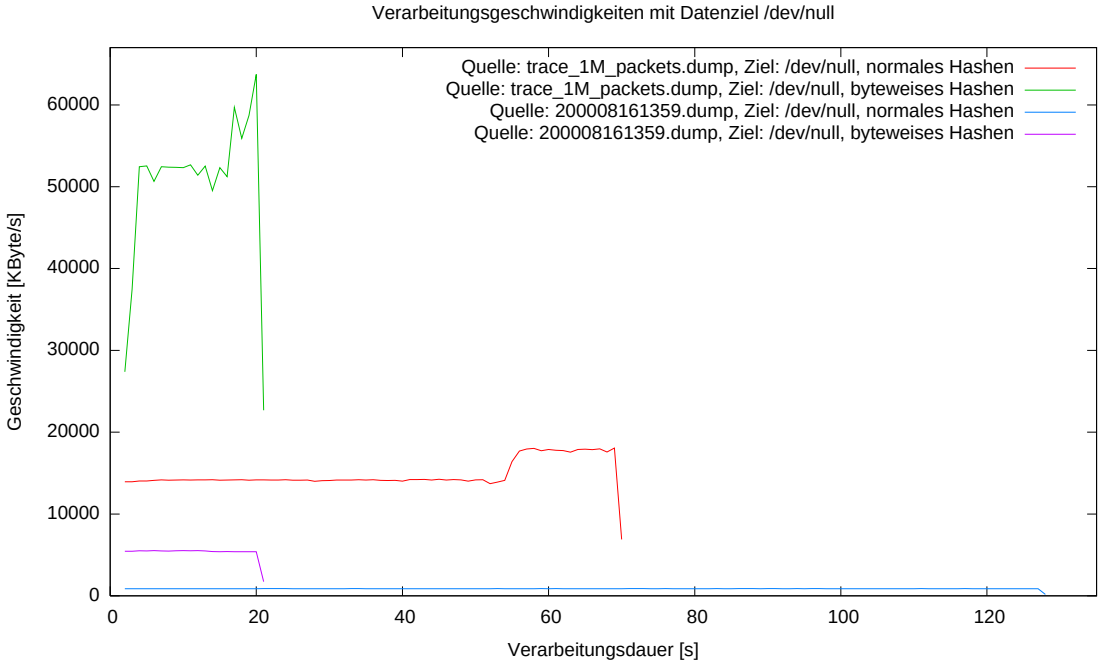

Abbildung 5.3: Verarbeitungsgeschwindigkeiten verschiedener Messungen mit Ziel `/dev/null`.

steht durch das CIDR-basierte Hashen ein gewisser Overhead, da für jede IP-Adresse der Binärbaum durchlaufen werden muss (siehe Abschnitt 5.4.2.2).

Um welchen Faktor sich die Verarbeitungsgeschwindigkeit beim Schreiben in eine Datei oder nach `/dev/null` erhöht, hängt direkt von der Paketgröße ab. So zeigt sich bei der Datei 200008161359.dump nur ein kleiner Performancesprung, wobei die Verarbeitungsgeschwindigkeit der Datei trace_1M_packets.dump maßgebend vom Datenziel abhängt. Der Grund dafür ist die fünffach größere Datenmenge der Datei trace_1M_packets.dump, verbunden mit der Tatsache, dass diese Datei weniger als die Hälfte der Pakete im Vergleich zur Datei 200008161359.dump enthält. Durch die geringere Anzahl an Paketen ist die Verarbeitungsgeschwindigkeit höher. Daher müsse Daten schneller auf die Festplatte geschrieben werden. Da jedoch die Paketgröße höher ist, müssen somit größere Datenblöcke auf die Festplatte geschrieben werden. Dadurch ist die Arbeit des Festplatte sehr viel höher, da einmal schneller Daten geschrieben werden müssen und auch die einzelnen Datenblöcke größer sind. Die Abbildungen 5.5 und 5.6 gruppieren jeweils die Messungen von byteweisem Hashen und ohne byteweises Hashen, um die Auswirkungen des Datenziels besser zu analysieren. Die Datei trace_1M_packets.dump zeigt durch Änderung der Datenquelle von einem Dateiziel nach `/dev/null`, laut Tabelle 5.3, eine Geschwindigkeitserhöhung von 147% bei Verwendung von byteweisem Hashen und eine Erhöhung von

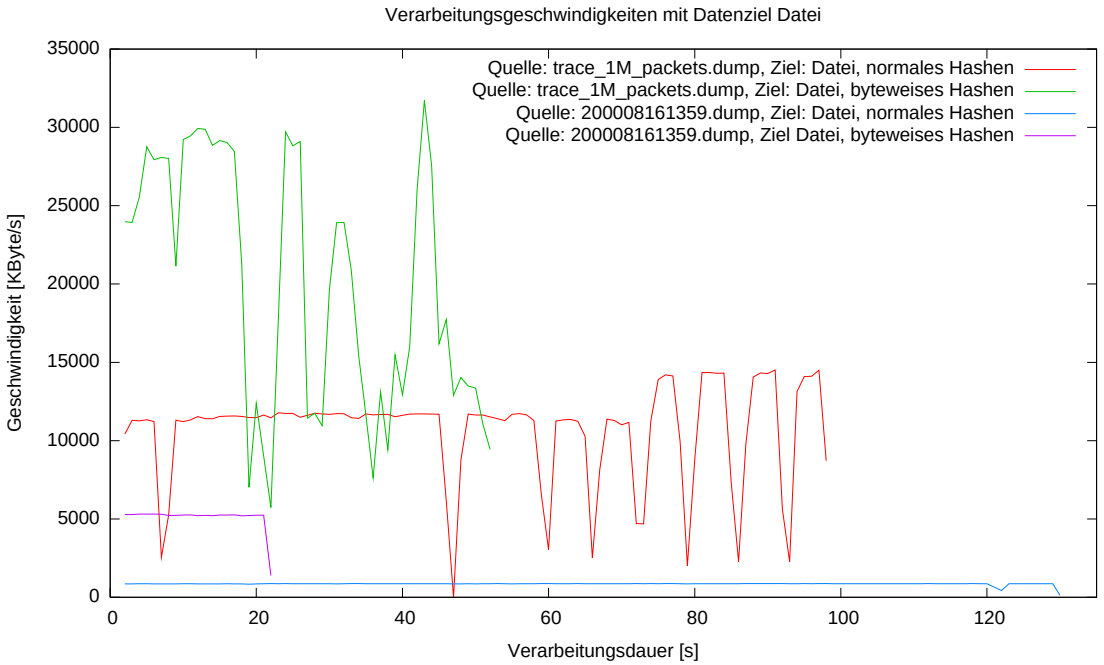

Abbildung 5.4: Verarbeitungsgeschwindigkeiten verschiedener Messungen mit Ziel Datei.

38% ohne byteweises Hashen. Im Gegensatz dazu erreicht die Verarbeitung der Datei 200008161359.dump mit byteweisem Hashen eine Erhöhung der durchschnittlichen Geschwindigkeit um 4% und ohne byteweises Hashen um 1%.

Abschließend wird für die Messungen mit einer Datei als Datenquelle die Verarbeitungsgeschwindigkeit pro Paket in Abbildung 5.7 betrachtet. Aufgrund der Prüfsummenberechnung dauert die Verarbeitung großer Pakete grundsätzlich länger als die Verarbeitung kurzer Pakete. Im Idealfall – beim Schreiben der Daten nach `/dev/null`, byteweisem Hashen und kleinen Paketen – wird laut Tabelle 5.3 eine durchschnittliche Verarbeitungsgeschwindigkeit von 0,011 Millisekunden pro Paket erreicht. Dies entspricht ungefähr 99 620 Paketen pro Sekunde und ist auch in Abbildung 5.8 zu sehen.

5.5.2 Messungen mit Tcpdump

Wie in Abschnitt 5.1.1 beschrieben, kann Pktanon direkt mit Tcpdump betrieben werden. Dabei werden die Daten, welche Tcpdump von der Netzwerkkarte abgreift, direkt als Eingabe für Pktanon verwendet. Pakete werden von TCPdump in die Standardausgabe geschrieben und von Pktanon aus der Standardeingabe gelesen. Solange ein Paket aus der Standardeingabe nicht abgeholt wird, verwirft TCPdump eingehende Pakete (siehe Abschnitt 5.4.4).
Der allgemeine Versuchsaufbau ist wie folgt: zwischen zwei Hosts A und B wird mit *Iperf* [Tiru] eine Datenübertragung hergestellt. Ein dritter Host S, auf welchem Pktanon betrieben wird, *snifft* die Pakete der Datenübertragung mit TCPdump. Die Pakete werden über Stdiobuffer zu Pktanon weitergeleitet. Das Programm Iperf wird mit dem UDP-Protokoll betrieben um eine möglichst hohe Datenrate zu erzielen.
Die Kommandozeile für die Ausführung des Iperf-Servers auf Host A lautet:

```
iperf -s -u -p 5000
```

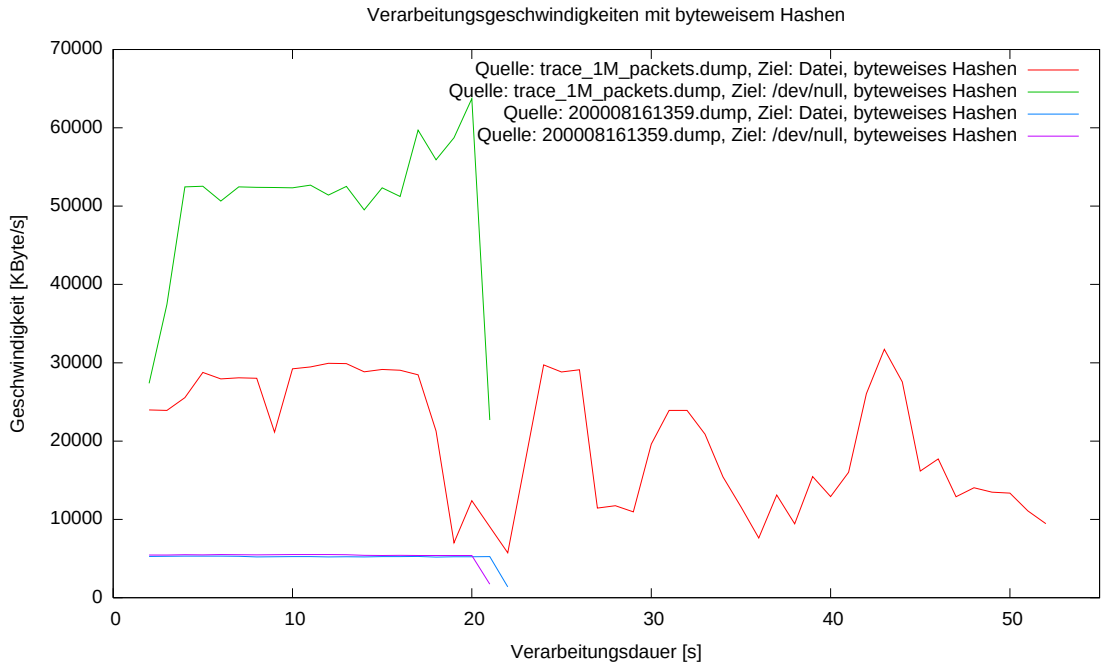

Abbildung 5.5: Verarbeitungsgeschwindigkeiten mit byteweisem Hashen und Dateiquelle.

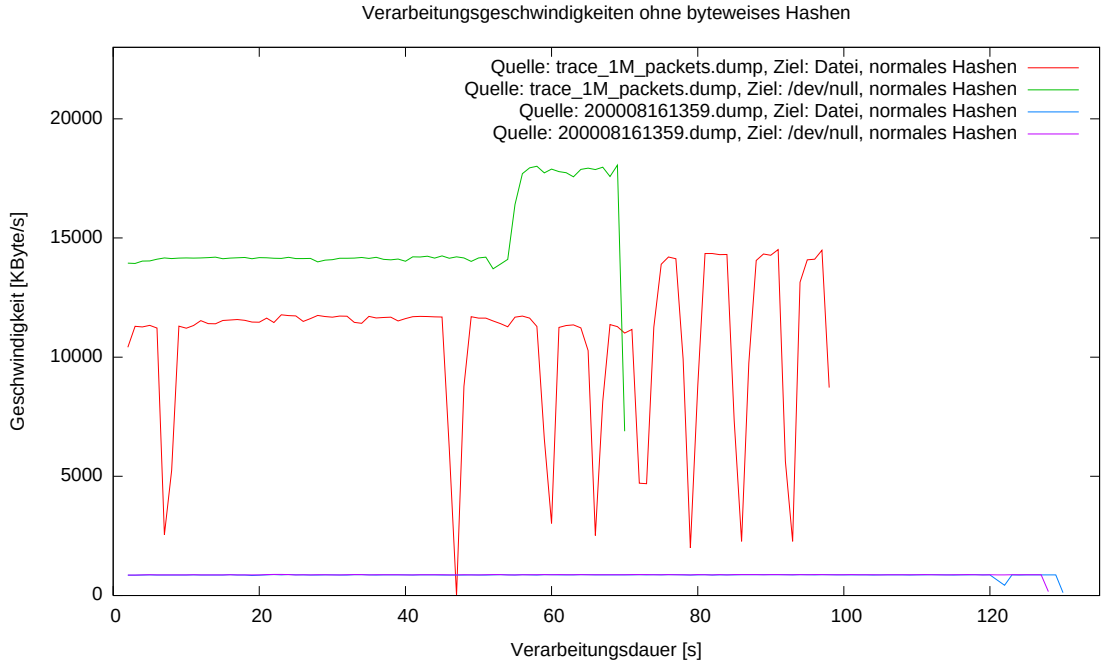

Abbildung 5.6: Verarbeitungsgeschwindigkeiten ohne byteweises Hashen und Dateiquelle.

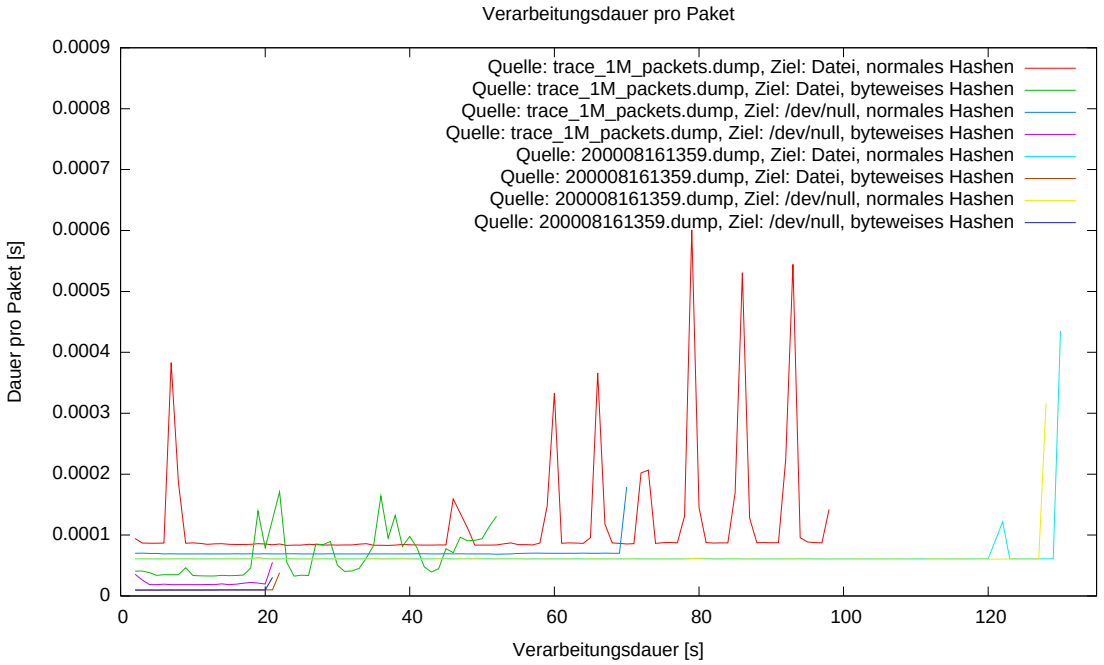

Abbildung 5.7: Verarbeitungsdauer pro Paket mit Dateiquelle

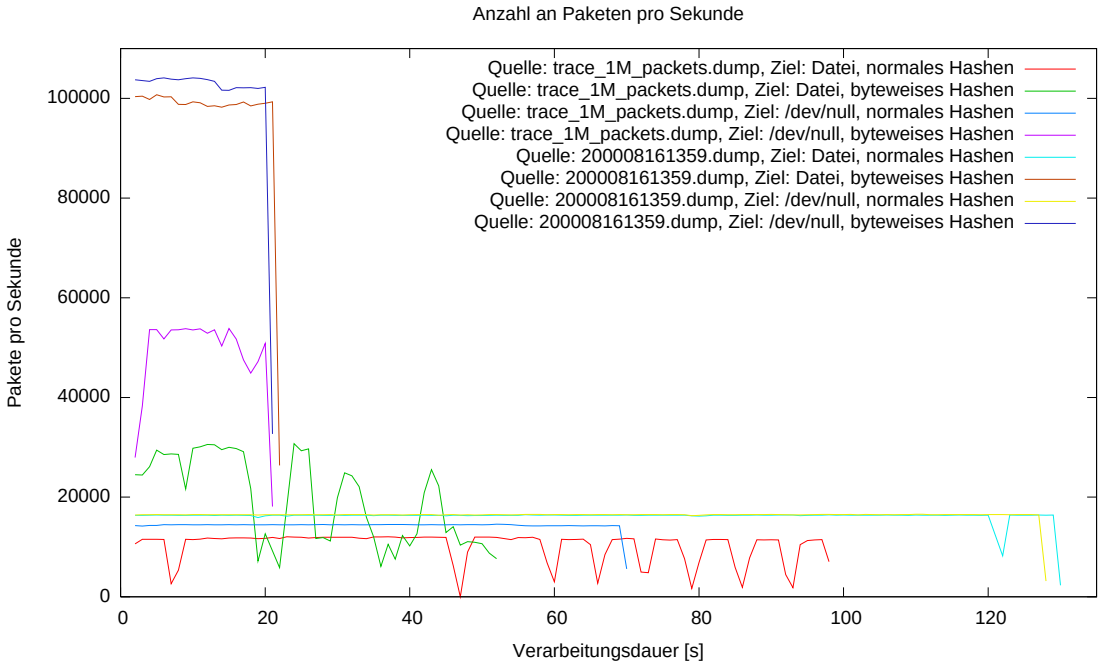

Abbildung 5.8: Anzahl an Paketen pro Sekunde bei Dateiquelle

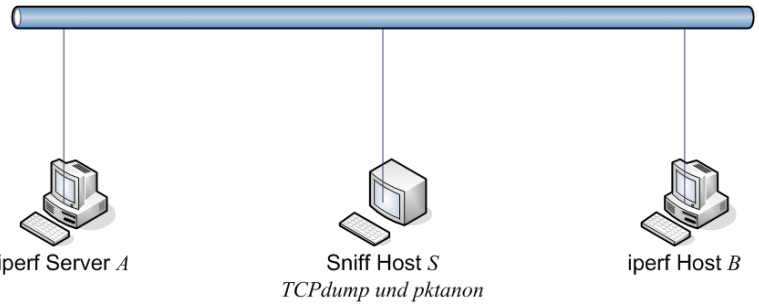

Abbildung 5.9: Versuchsaufbau zur Online-Anonymisierung

Dabei wird mit der Option `-s` der Server-Mode aktiviert, die Option `-u` wird zur Verwendung von UDP benutzt und die Option `-p` definiert den Port, auf welchem der Server auf eine Verbindung wartet.
Die Kommandozeile für die Ausführung des Iperf-Clients auf Host B lautet (hier beispielhaft mit einer Übertragungsgeschwindigkeit von 50 MBit/s):

```
iperf -u -c A -p 5000 -b 50M
```

Die Option `-u` wählt UDP als Transportprotokoll, der Host, zu welchem der Client sich verbinden soll, wird mit der Option `-c` gesetzt, der Port, zu welchem sich der Client auf dem Server verbinden soll, wird durch die Option `-p` gewählt und die Option `-b` definiert die Übertragungsgeschwindigkeit.

TCPdump wird mit den Parametern `-s 0` und `-U` betrieben (siehe auch Abschnitt 5.1.1), um den kompletten Datenstrom aufzunehmen und die Weitergabe von einzelnen Paketen zu Pktanon direkt zu veranlassen, anstatt zu warten bis der Ausgabepuffer gefüllt ist. Abbildung 5.9 zeigt den Versuchsaufbau.

5.5.3 Verlustmessung

Um den Paketverlust bei verschiedenen Datenraten zu analysieren, wurde der prozentuale Anteil der Pakete gemessen, welche TCPdump verwirft. Dies ist eine Metrik für die Performance von Pktanon. Bei diesen Messungen wird grundsätzlich in eine Datei geschrieben und wahlweise der Zwischenspeicher Stdiobuffer verwendet (siehe Abschnitt 5.4.4). Abbildung 5.10 zeigt den prozentualen Paketverlust bei verschiedenen Übertragungsgeschwindigkeiten. Die Datenübertragung dauerte bei diesen Messungen jeweils 120 Sekunden. Wie man sieht, sind beim byteweisen Hashen mit Zwischenspeichern der Ergebnisse grundsätzlich weniger Paketverluste als beim normalen Hashen. Dies ist auf Grund der vorangegangenen Messungen in Abschnitt 5.5.1 nicht weiter verwunderlich, da das byteweise Hashen eine höhere Verarbeitungsgeschwindigkeit erlaubt, wodurch letztendlich weniger Pakete verworfen werden. Erkennbar ist vor allem der niedrigere Paketverlust bei Verwendung von Stdiobuffer. Bei höheren Übertragungsgeschwindigkeiten bringt dieser Mechanismus keinen Vorteil, da der Zwischenspeicher dann komplett gefüllt ist und damit wieder die normalen Paketverluste auftreten. Die Verwendung von Stdiobuffer verschlechtert in diesem Fall die Performance eher, da Stdiobuffer selbst Prozessorleistung verbraucht. Bei niedrigeren Übertragungsgeschwindigkeiten und Paketverlusten bis

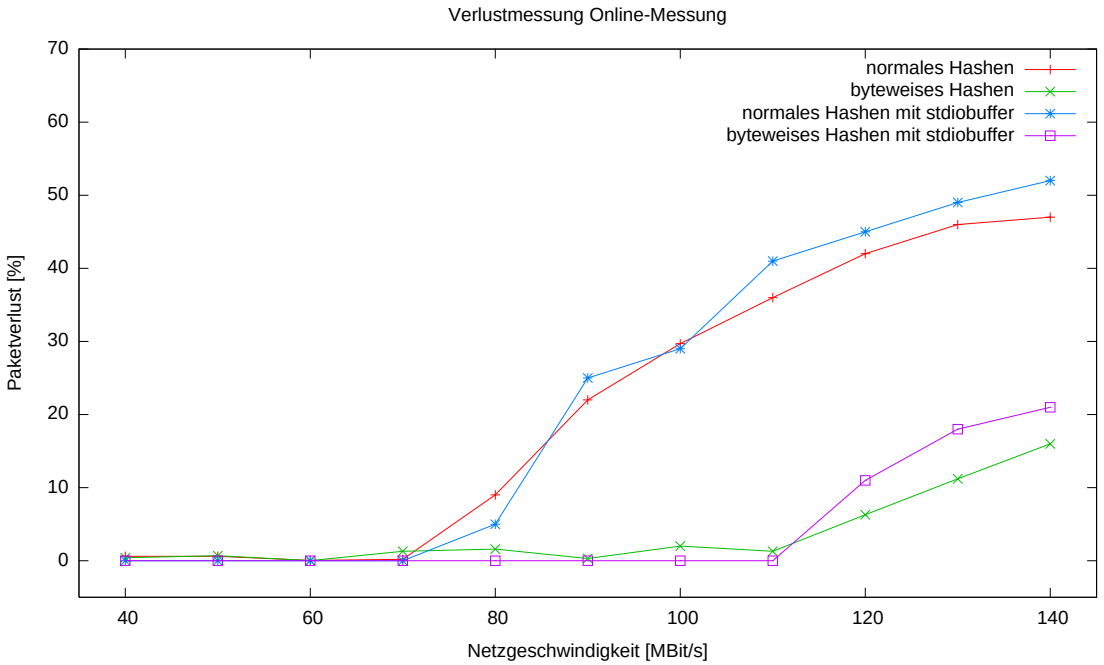

Abbildung 5.10: Paketverluste bei verschiedenen Betriebsarten und Übertragungsgeschwindigkeiten

zu 10% ist diese Methode jedoch effektiv und kann die Paketverluste auf Werte gegen 0% senken.
Wie aus Abbildung 5.10 abzulesen ist, ist bei Verwendung von byteweisem Hashen das Online-Anonymisieren bei einer Datengeschwindigkeit von 110 MBit/s ohne Paketverlust möglich. Mit normalem Hashen ist ein Online-Anonymisieren bei bis zu 70 MBit/s ohne Paketverlust möglich.

5.5.4 Messung der Verarbeitungsgeschwindigkeit

Durch die eingebaute Messengine (siehe Abschnitt 5.1) ist es möglich, die Verarbeitungsgeschwindigkeit von Pktanon zu messen. Da sich die Verwendung des Zwischenpuffer Stdiobuffer in Abschnitt 5.5.3 als effektiv erwiesen hat, wurde dieser auch hier verwendet. Zur Messung wurde eine Übertragungsgeschwindigkeit von 95 MBit/s zwischen Host A und B verwendet. Dies war die höchstmögliche Übertragungsgeschwindigkeit, die zwischen den beiden Hosts auf einer 100 MBit/s Leitung erreicht wurde. Wie in Abbildung 5.11 zu sehen ist, erreicht Pktanon mit der Verwendung des byteweisen Hashens genau die Verarbeitungsgeschwindigkeit, welche als Übertragungsgeschwindigkeit zwischen Host A und B gewählt wurde. Bei Verwendung des normalen Hashens ist zu sehen, dass die Verarbeitungsgeschwindigkeit sehr schwankt. Grund dafür ist die hohe Beanspruchung der CPU durch die vielen Hashoperationen, welche bei jedem Paket ausgeführt werden müssen. Bei Verwendung des byteweisen Hashens hilft das Zwischenspeichern der gehashten Bytes, dass insgesamt nur wenige Hashoperationen durchgeführt werden müssen.

Um Messungen mit realem Verkehr durchzuführen wurde in Kooperation mit dem Rechenzentrum der Universität Karlsruhe Datenverkehr eines realen VLAN über einen Mirror-Port umgeleitet und mit Pktanon aufgenommen. Die Messungen sind in Abbildung 5.12 und 5.13 zu sehen.

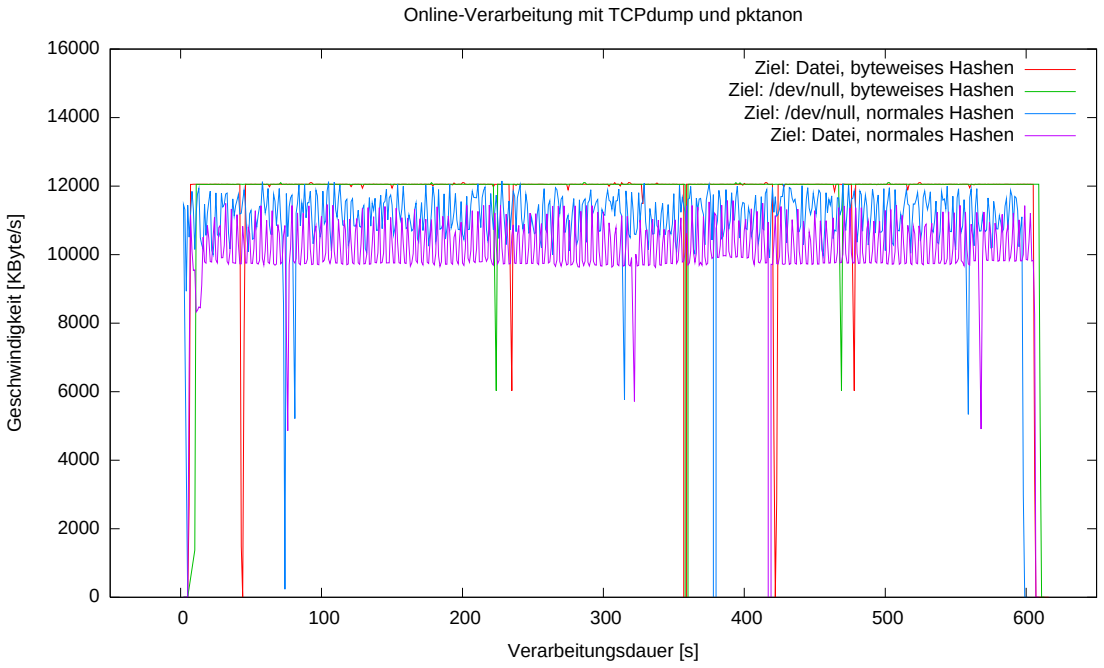

Abbildung 5.11: Verarbeitungsgeschwindigkeiten bei 95 MBit/s

Abbildung 5.12 zeigt, dass bei dieser ersten Messung der Datenverkehr sehr schwankte, was für realen Verkehr unter anderem charakteristisch ist. Die Paketverluste bei diesen Messungen lagen immer unter 1%. Daraus kann man schließen, dass Pktanon nicht durch das Datenaufkommen ausgelastet wurde.
Abbildung 5.13 zeigt die Verarbeitungsgeschwindigkeiten der zweiten Messung. Bei dieser wurde aus einer hohen Datenrate zwischen 200 und 500 MBit/s ein genau 100 MBit/s starker Strom ausgewählt. Wie zu sehen ist, erreichte die Verarbeitung mit byteweisem Hashen und Schreiben der Daten nach `/dev/null` genau diese Geschwindigkeitsmarke. Beim byteweisen Hashen und Schreiben in eine Datei konnte die Festplatte nicht schnell genug die Daten verarbeiten, daher sind die Verarbeitungsgeschwindigkeiten dort geringer. Der Zwischenspeicher Stdiobuffer wurde bei dieser Messung nicht verwendet. Wie in Abschnitt 5.4.4 beschrieben, hätte bei Verwendung von Stdiobuffer die Verlustrate gesenkt werden können.

5.6 Zusammenfassung

Die Anonymisierung mit Pktanon zeigt sich sowohl bei Offline- wie auch Onlineanonymisierung effektiv und flexibel. Verschiedene Optimierungen verbessern die Performance teilweise erheblich. Verschiedene Hashverfahren stehen zur Verfügung, welche unterschiedliche Eigenschaften bezüglich Geschwindigkeit, Sicherheit und Erhaltung der Subnetzverteilung besitzen. Durch das generische Framework ist Sicherheit gewährleistet und Erweiterungen sind leicht möglich.

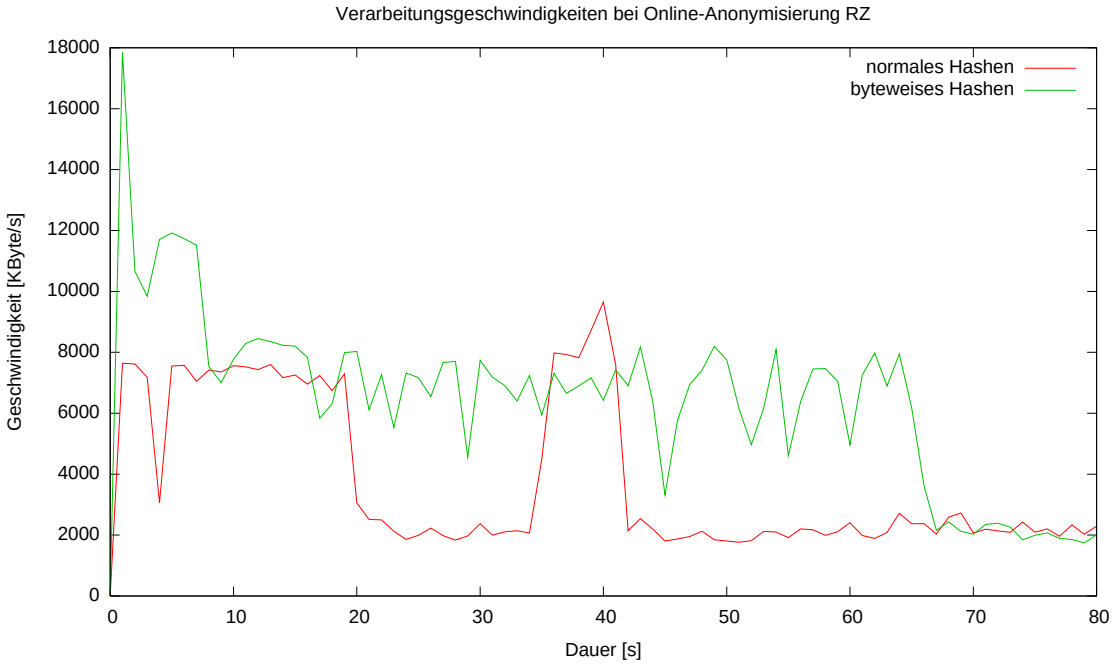

Abbildung 5.12: Verarbeitungsgeschwindigkeiten mit realem Netzverkehr

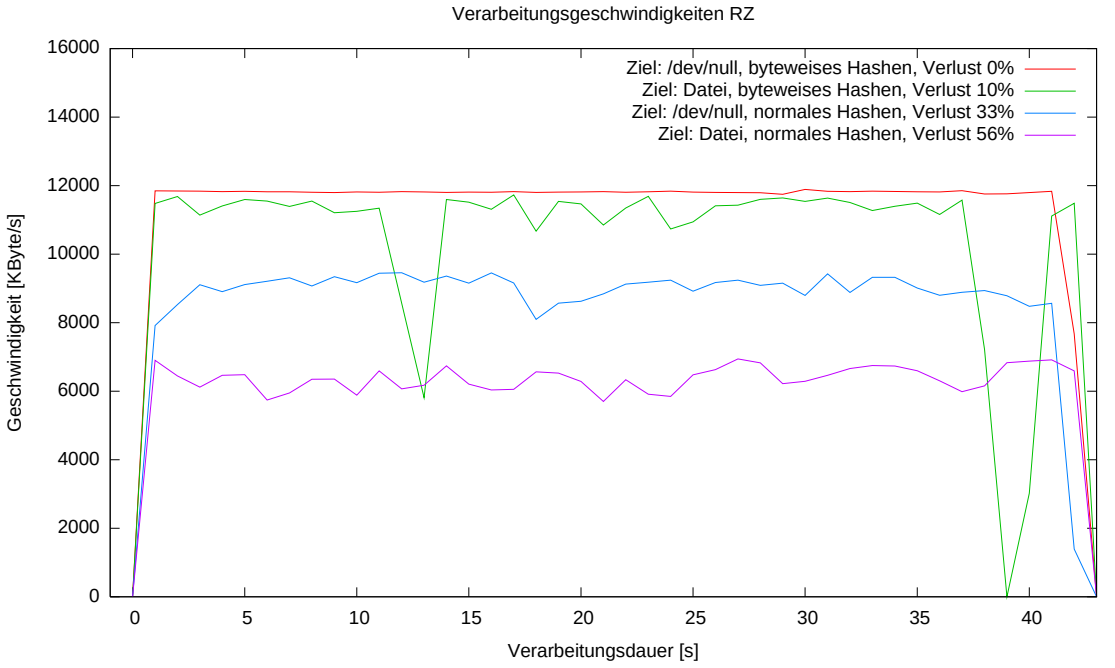

Abbildung 5.13: Verarbeitungsgeschwindigkeiten mit realem Netzverkehr

6. Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein Framework zur Anonymisierung von Netzdaten konzipiert, implementiert und evaluiert. Es wurde gezeigt, dass der generische Aufbau des Frameworks und die aufgezeigte Handhabung der Netzdaten zu einer sicheren und schnellen Anonymisierung führen. Durch die verkettete Handhabung von Paketen ist es möglich, beliebige Verschachtelungen von Protokollen zu unterstützen und neue Protokolle schnell und einfach zu implementieren und zu integrieren. Durch das große Maß an Flexibilität und hohen Verarbeitungsgeschwindigkeiten eignet sich das Framework somit für den realen Betrieb. Möglichkeiten wie Online-Anonymisierung minimieren die datenschutzrechtlichen Bedenken und gewährleisten zusätzliche Sicherheit.
Um die Anwendungsgebiete von Pktanon noch weiter auszubauen, sind folgende Ansatzpunkte zur Weiterentwicklung denkbar. Zum einen ist die Erhöhung der Verarbeitungsgeschwindigkeit in die Bereiche von Gigabit sehr interessant. Dadurch kann Pktanon an zentralen Stellen von Netzen eingesetzt werden. An diesen Punkten im Netzinneren entsteht sehr viel ungefilterter Verkehr und eine sehr große Zahl an verschiedenen Protokollen ist dort zu beobachten. Somit wäre es möglich große Datenmengen aufzunehmen, welche komplett ungefiltert sind und sehr interessante Ansätze für Analysen geben. Um dieses Ziel zu erreichen, müssen interne Abläufe – unter anderem das Speichermanagement – weiter optimiert werden. Durch Einbauen von Optionen wie das Nichtberechnen von Prüfsummen oder Auslagerung der Prüfsummen- und Hashberechnungen in Hardware sind Erhöhungen der Verarbeitungsgeschwindigkeit erreichbar. Weiterhin stellt die Unterstützung von Mehrprozessorsystemen und Multicore-Prozessoren einen Schritt hin zu höheren Verarbeitungsgeschwindigkeiten dar.
Des Weiteren ist auch die Verwendung einer größeren Zahl an Protokollen interessant. Hier ist zu untersuchen, ob eine Protokollbeschreibungssprache entwickelt werden kann, mit welcher Protokolle definiert werden können. Somit kann verhindert werden, dass jedes Protokoll einzeln in Pktanon implementiert werden muss, und andere Anwendungen können von einer solchen Protokollbeschreibung profitieren.
Das Pktanon Framework unterstützt bereits eine Menge an Anonymisierungsprimitiven. Hier können weitere Anonymisierungsprimitiven eingebaut werden um einen

höheren Grad an Konfigurierbarkeit zu erreichen.
Um Anonymisierungsprimitiven den Protokollattributen zuzuordnen, ist es von praktischem Interesse Konfigurationsprofile zu verwenden, mit welchen sich diese Bindungen einfach modellieren lassen. Auf diese Weise können aus juristischer Sicht Vertrauens- und Sicherheitsverhältnisse in den Konfigurationsprofilen modelliert werden. Benutzer können dann auf eines der Profile zurückgreifen, wodurch Fehler bei der Konfiguration vermieden werden, eine höhere Benutzerfreundlichkeit erreicht wird und juristische Absicherung gegeben ist.

Literatur

[8023] IEEE 802.3 Working Group. http://grouper.ieee.org/groups/802/3/.

[Adam04] William Yurcik Adam J Slagell. Sharing Computer Network Logs for Security and Privacy: A Motivation for New Methodologies of Anonymization, 2004. http://slagell.com/downloads/slagell05b.pdf.

[bund] Bundesdatenschutzgesetz. http://www.gesetze-im-internet.de.

[cisc] Cisco Systems, Inc. http://www.cisco.com.

[FIPS02] FIPS. *Secure Hash Standard*, Aug 2002. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf.

[Glog] Wolfram Gloger. ptmalloc. http://www.malloc.de.

[hone] The Honeypot Project. http://project.honeynet.org.

[ita] The Internet Traffic Archive. http://ita.ee.lbl.gov/index.html.

[J. F] M. H. Ammar J. Fan, J. Xu. Crypto-PAn, Cryptography-based Prefix-preserving Anonymization. http://www-static.cc.gatech.edu/computing/Telecomm/cryptopan/.

[Kenj] Akira Kato Kenjiro Cho, Koushirou Mitsuya. Traffic Data Repository at the WIDE Project. S. 263–270. http://www.sfc.wide.ad.jp/ mitsuya/PAPERS/freenix2000.pdf.

[Labo] Peter Laborge. WMF flaw sold for $4,000. Securityfocus. http://www.securityfocus.com/brief/126.

[Lea] Doug Lea. A Memory Allocator. http://g.oswego.edu/dl/html/malloc.html.

[Loza] Boris Loza. TCP Timestamp To count Hosts behind NAT, Phrack Magazin. http://www.phrack.org.

[MAWI] MAWI-Working-Group. Wide-tcpdpriv. http://tracer.csl.sony.co.jp/mawi/.

[Mins] Greg Minshall. Tcpdpriv. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html.

[Mogu02] Jeffrey Mogul. Trace anonymization misses the point. 2002. http://www2002.org/presentations/mogul-n.pdf.

[Patr04] Vitaly Shmatikov Patrick Lincoln, Phillip A. Porras. Privacy-Preserving Sharing and Correlation of Security Alerts. In *USENIX Security Symposium*, 2004, S. 239–254. http://citeseer.ist.psu.edu/context/2675098/0.

[Paxs] Vern Paxson. Sanitize. http://ita.ee.lbl.gov/html/contrib/sanitize.html.

[Perk96] C. Perkins. IP Encapsulation within IP. RFC 2003 (Proposed Standard), Oct 1996. http://www.ietf.org/rfc/rfc2003.txt.

[Peuh01] Markus Peuhkuri. A method to compress and anonymize packet traces. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, New York, NY, USA, 2001. ACM Press, S. 257–261. http://www.imconf.net/imw-2001/imw2001-papers/32.pdf.

[Post80] J. Postel. User Datagram Protocol. RFC 768 (Standard), Aug 1980. http://www.ietf.org/rfc/rfc768.txt.

[Post81a] J. Postel. Internet Control Message Protocol. RFC 792 (Standard), Sep 1981. http://www.ietf.org/rfc/rfc792.txt.

[Post81b] J. Postel. Internet Protocol. RFC 791 (Standard), Sep 1981. http://www.ietf.org/rfc/rfc791.txt.

[Post81c] J. Postel. Transmission Control Protocol. RFC 793 (Standard), Sep 1981. http://www.ietf.org/rfc/rfc793.txt.

[R. S] J. Heidemann R. Sinha, C. Papadopoulos. Internet Packet Size Distributions: Some Observations. http://netweb.usc.edu/ rsinha/pkt-sizes/.

[Reid] Steve Reid. Public Domain SHA1 implementation. http://www.mirrors.wiretapped.net/security/cryptography/hashes/sha1/sha1.c.

[Roes] Martin Roesch. Snort Intrusion Detection System. http://www.snort.org.

[Ruom03] Vern Paxson Ruoming Pang. A high-level programming environment for packet trace anonymization and transformation. In *SIGCOMM 03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, 2003. ACM Press, S. 339–351. http://doi.acm.org/10.1145/863955.863994.

[Schu] Henning Schulzrinne. Trace Sammlung der University of Columbia. http://www.cs.columbia.edu/ hgs/internet/traces.html.

[Secu] Securityfocus. SecurityFocus Vulnerability Database. http://www.securityfocus.com/vulnerabilities.

[Simp95] W. Simpson. IP in IP Tunneling. RFC 1853 (Informational), Oct 1995. http://www.ietf.org/rfc/rfc1853.txt.

[Syma] Symantec. Symantec DeepSight Analyzer. http://analyzer.symantec.com/.

[tele] Telekommunikationsgesetz. http://www.gesetze-im-internet.de.

[Tiru] Dugan Tirumala, Qin. Iperf. http://dast.nlanr.net/Projects/Iperf/.

[Touc97] J. Touch. TCP Control Block Interdependence. RFC 2140 (Informational), Apr 1997. http://www.ietf.org/rfc/rfc2140.txt.

[XFAM01] J. Xu, J. Fan, M. Ammar und S. Moon. On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization. In *ACM SIGCOMM Internet Measurement Workshop (IMW)*, San Francisco, Nov 2001. http://www.imconf.net/imw-2001/imw2001-papers/69.pdf.

[zero] Companies adapt to a zero day world. http://www.securityfocus.com/news/9100.