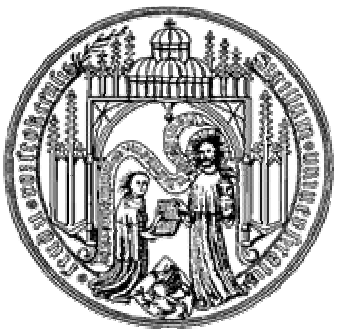
Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Nachrichtentechnik und Informationselektronik



Großer Beleg

Erweiterung einer bestehenden Softwareanwendung für die floworientierte Auswertung von IP Datenverkehr

cand. Ing. Christian Manthey

Betreuer: Dr.-Ing. Hans-Dietrich Melzer
Dipl.-Ing. Thomas Kessler

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

A

ACK	Acknowledge
AN	Acknowledge Number
API	Advanced Programming Interface

C

CSV	Comma Separated Values

D

DBS	Datenbanksystem

F

FIN	Finish

H

HTTP	Hypertext Transport Protocol

I

IP	Internet Protocol
ISN	Initial Sequence Number

P

PAR	Positive Acknowledge with Retransmission

R

RFC	Request for Comment

S

SN	Sequence Number
SQL	Structured Query Language
STD	State Transition Diagram
SYN	Synchronization

T

TCP	Transmission Control Protocol

U

UDP	User Datagram Protocol

W

WWW	World Wide Web

1 Motivation

Es gibt viele Gründe, warum in einigen Anwendungen die floworientierte Betrachtungsweise gegenüber einer rein quantitativen statistischen Analyse von Verkehrsdaten sinnvoller ist. Darüber wurde in [1] diskutiert. Neben der Frage, welches Modell eines Verkehrsflusses für IP-Daten man der Untersuchung zu Grunde legt, ist aber auch von Bedeutung, wie man diese erzeugten Daten sinnvoll und effektiv weiterverarbeitet. Die Antwort auf diese Frage ergibt sich einerseits aus den technischen Möglichkeiten des Systems, vielmehr jedoch daraus, welche Schlussfolgerungen man aus den erzeugten Flowdaten ziehen und welche Aussagen man über die bewerteten Verkehrsdaten treffen möchte.

Die durch *recordsneaker* in [1] erzeugten Ausgabedateien, ob nun in Tabellen- oder CSV-Form, sind für eine weitere Bearbeitung jedoch nur bedingt geeignet. Einerseits gestaltet sich die Suche in solchen Dateien als ineffektiv, umständlich und langsam. Diese Tatsache verschärft sich zunehmend mit der Erzeugung größerer Ausgabedateien. Der Dateizugriff und eine Suche hat mit einem Anstieg der Datenmenge in den Dateien eine überproportional wachsende Bearbeitungszeit zur Folge. Ist zudem eine vollständige Auswertung der Daten in Echtzeit gefordert, so ist die Arbeit von *recordsneaker* bei gleichzeitiger Analyse der Flow-Dateien nur begrenzt möglich.

Hier bietet sich als Ausweg an, die Flowdaten über ein Ausgabemodul an eine MySQL-Datenbank zu übergeben. Ein MySQL-Server kann als eigene Instanz

sogar auf einem anderen Rechnersystem laufen und würde die Arbeit des DAG-Messsystems und *recordsneaker* nicht behindern. Gleichzeitig bietet es die uneingeschränkte Möglichkeit, die Flowdaten in der Datenbank zu bearbeiten und durchsuchen. Dies ist bei Bedarf auch in Echtzeit möglich und wird nur von den Ressourcen des Systems, auf dem der MySQL-Server arbeitet und von dessen Konfiguration beeinflusst. Daher bietet es sich an, in dem im Rahmen von [1] erstellten Programm *recordsneaker* zusätzlich ein Ausgabemodul für eine MySQL-Datenbank zu integrieren. Diese Ausgabemöglichkeit sollte sich dem bestehenden System anpassen und dieses möglichst nicht beeinflussen.

Im Ausblick von [1] wurde weiterhin geschildert, dass im Rahmen einer weiteren Analyse von Verkehrsdaten auch Paketinformationen aus höheren Schichten von Interesse sein könnten. Sofern dies aus den durch das Messsystem aufgezeichneten Daten hervorgeht, sollten daher zusätzlich zu den schon generierten Flowdaten Payloads von Paketen ausgelesen werden, aus denen später Informationen über die die IP-Verbindung benutzenden Anwendungen gewonnen werden können. Nicht jede Payload ist dabei von Bedeutung, am aussagekräftigsten sind dabei jene Payloads, die am Anfang einer solchen Verbindung auftreten. Verbindungsaufbauten sind in der Regel durch eine Anfrage (Request, Call) an eine Gegenstelle charakterisiert. Aus diesen Anfragen lässt sich am ehesten auf die Anwendung (z.B. eine HTTP-Verbindung eines Webbrowsers mit einem Webserver) schließen.

Das Erkennen eines Verbindungsanfangs ist jedoch zuverlässig nur bei TCP möglich. Dort findet ein nachvollziehbarer Initialisierungsvorgang beim Aufbau der Verbindung statt, der sich anhand von Informationen in den Headern der TCP-Segmente erkennen lässt. Ein solcher Mechanismus existiert für UDP-Verbindungen nicht, eine Erkennung eines Verbindungsstarts ist im Rahmen des Messvorganges und der Auswertung durch *recordsneaker* nicht ohne weiteres möglich. Das Auslesen einer beliebigen Payload innerhalb des Flows

macht wenig Sinn, aus dieser wird sich in den seltensten Fällen zuverlässig ein Anhaltspunkt über die kommunizierende Anwendung ergeben.

Der Schwerpunkt sollte daher auf die TCP-Verbindung gelegt werden. Hier ist die erste Payload eines TCP-Segments nach einem erfolgreichen Verbindungsaufbau auszulesen und den vorhandenen Flowdaten hinzuzufügen.

Weiterhin sollten an *recordsneaker* mögliche Verbesserungen der Programm-Ablaufgeschwindigkeit vorgenommen werden. Dies ist insbesondere durch Verringerung von Zwischenspeicher-Vorgängen zu erreichen.

2 Vorüberlegungen

Zwei wichtige Fragestellungen sind zu klären, bevor die beiden Hauptaufgaben, die Protokollierung der Payload des ersten TCP-Paketes nach erfolgreichem Handshaking und die MySQL-Datenbankintegration, im Programm umgesetzt werden können.

Protokollierung des ersten TCP-Paketes nach erfolgreichem Handshaking

Warum soll nur die Payload der TCP basierten Flows ausgewertet werden? Natürlich sind für eine Analyse auch UDP Flows von Interesse. Jedoch ist davon auszugehen, dass der interpretierbare Teil eines Paketes regelmäßig zu Beginn einer neuen Verbindung respektive eines Flows zu finden ist. Dort wird im Allgemeinen die Anfrage eines Knotens an eine Gegenstelle in Form eines „Requests“, „Calls“ oder ähnlich geartetem Kommando gestellt. Aus den Details dieses Kommandos, beispielsweise ein HTTP-Request „*GET*“ eines Browsers an einen Server, lassen sich Schlussfolgerungen auf die den Flow benutzende Anwendung (in diesem Fall das in höheren Schichten angesiedelte WWW-Protokoll HTTP) ziehen, die über eine reine Interpretation anhand der Portnummern (z.B. Anfrage an Port 80 eines Servers => i.A. WWW) hinausgehen. UDP ist jedoch ein verbindungsloses Protokoll, welches keine Verkehrssteuerung in der Transportschicht durchführt. Das Erkennen des Anfangs eines UDP Flows würde mit den in [1] beschriebenen und verwendeten Definitionen des Verkehrsflusses ohne zusätzliche Informationen

und Aufwand nicht möglich sein. Netzwerkanwendungen, welche UDP als Transportprotokoll verwenden, führen die für eine erfolgreiche Kommunikation notwendigen Mechanismen im Allgemeinen in höheren Schichten durch.

Für die TCP-Verbindung ist zu klären, wie die Verkehrssteuerung funktioniert. Darauf wird im Kapitel 3 eingegangen. Im Kapitel 5 wird beschrieben, eine sichere Erkennung eines erfolgreichen Verbindungsstarts in *recordsneaker* zu implementieren ist.

MySQL-Datenbankintegration

Um die Möglichkeiten der Auswertung der erzeugten Flowdaten zu erhöhen, ist die Ausgabe in einer Datenbank von Interesse. Erzeugte Dateien (z.B. CSV) eignen sich nur bedingt für eine weitergehende Analyse. Die Suche nach Flows mit bestimmten Eigenschaften in Dateien wäre sehr schwierig und zudem schwerfällig. Das ist insbesondere dann der Fall, wenn die Menge der Verkehrsdaten rapide anwächst. Ein Datenbanksystem hingegen erlaubt die schnelle Suche in Datenbeständen und bietet Mechanismen für den Umgang mit größeren Datenmengen. Auch Sortiervorgänge lassen sich wesentlich leichter bewerkstelligen als es mit den erzeugten Ausgabefiles der Fall wäre. Hinzu kommt die Tatsache, dass im Fall einer gleichzeitigen parallelen Bearbeitung bzw. Analyse der Datenbankinhalte ein DBS ein Transaktionsmanagement anbietet. Das vermeidet Kollisionen bei eventuell gleichzeitigem Datenbank-Zugriff von *recordsneaker* und einem anderen Client, der zur Auswertung dient.

Als Datenbanksystem eignet sich das frei verfügbare relationale Datenbanksystem **MySQL** ([3]). Es ist kostenlos, einfach zu installieren und die Kommunikation zwischen einem Daten erzeugenden Programm (hier *recordsneaker*) und dem DBS ist mit wenig Aufwand und dank einer

standardisierten und gut dokumentieren API problemlos möglich. Die Möglichkeiten von MySQL sind vielfältig, diese Stärken werden jedoch wenig in der Kommunikation mit *recordsneaker*, sondern viel mehr bei der späteren Analyse ausgespielt werden. MySQL zeichnet sich zudem durch eine hohe Stabilität und geringe Anforderungen an die Systemressourcen aus, so dass möglichst viel davon für die Nutzdaten zur Verfügung steht.

3 Verbindungssteuerung mit dem TCP-Protokoll

An dieser Stelle soll das TCP-Protokoll ein wenig näher beleuchtet werden. Der Schwerpunkt soll dabei auf der Flusssteuerung liegen, da diese für die Erkennung des ersten „interessanten“ Paketes von wichtiger Bedeutung ist. Eine ausführliche Beschreibung der gesamten Protokollsuite findet sich in [2].

TCP (Transmission Control Protocol) bietet im Gegensatz zu UDP eine Kommunikation an, mit der der Anwendung ein zuverlässiger End-to-End Dienst geboten wird. Das verbindungsorientierte TCP setzt auf IP auf und wird auch, wegen seiner Flusskontrolle auf Oktettbasis, als *reliable stream transport service* bezeichnet. TCP stellt die folgenden Dienste zur Verfügung:

- **End-to-End Kontrolle**

  Das TCP stellt eine korrekte Datenübertragung durch einen positiven Quittierungsmechanismus sicher. Verlorengegangene Daten werden nochmals übertragen.

- **Multiplexmechanismus**

  Über die Portadressierung erfolgt ein Multiplexen zu den Anwendungen. Weiterhin wird eine Verbindung durch ihre beiden Sockets (IP-Adresse und Portnummer) identifiziert; deshalb können zwischen zwei Stationen parallele Verbindungen zu identischen (meist well known-) Sockets bestehen.

- **Verbindungsmanagement**

Der gesicherte Aufbau einer Verbindung (3-Wege-Handshake), die gesicherte Aufrechterhaltung der (virtuellen) Verbindung während des gesamten Informationsaustauschs sowie ein gesicherter Abbau (bzw. ein erzwungener Abbruch) der Verbindung machen das Verbindungsmanagement aus.

- **Datentransport**

Der Datentransport kann nach einem erfolgreichen Aufbau der Verbindung bidirektional (vollduplex) erfolgen.

- **Flusskontrolle**

Der Datenstrom wird durch Sequenz- und Bestätigungsnummern auf Oktettbasis zuverlässig übertragen. Mit einem Windowing-Mechanismus wird ein Überlauf beim Empfänger verhindert.

- **Zeitüberwachung der Verbindung**

Übertragene Daten müssen innerhalb einer definierten Zeit vom Empfänger quittiert werden. Erfolgt diese Quittierung nicht, werden die Daten erneut übertragen. (retransmission mechanism)

- **Reihenfolge**

Client und Server tauschen vollkommen transparent Datenströme aus. TCP garantiert, dass die Daten in derselben Reihenfolge den Empfängerprozess erreichen wie sie vom Senderprozess verschickt wurden.

- **Spezialfunktionen**

Über die beiden Spezialfunktionen *PUSH* und *URGENT* können Vorrangdaten (meist Befehle, die die Verbindung selbst betreffen), gekennzeichnet werden, *PUSH* erzwingt die sofortige Datenübergabe an die nächst höhere bzw. nächst niedrigere Schicht. Ein *URGENT* Segment zeigt dem Empfänger dringende Daten an (out of band); es liegt jedoch am Empfänger, wie diese Information genutzt wird.

- **Fehlerbehandlung**

Wird ein empfangenes Segment als fehlerhaft erkannt, wird es verworfen. Die Daten werden daher nicht positiv quittiert; es erfolgt eine automatische *retransmission*.

TCP ist im Gegensatz zu UDP ein zustandsabhängiges Protokoll. Das heißt, dass die Reaktion auf eine neue zwischen den Kommunikationspartnern ausgetauschte Nachricht nicht nur von deren Inhalt, sondern auch von der vorangegangen „Geschichte“ des Datenaustausches zwischen den beiden abhängig ist. Das ermöglicht einerseits die oben genannten Dienste, erhöht aber gleichzeitig die Komplexität der Auswertung des Kommunikationszustandes. Dieser muss bei der programmtechnischen Umsetzung Rechnung getragen werden.

Abbildung 3.1 zeigt den Zustandsübergangsgraphen (*state transition diagram*) mit den vereinfachten Zuständen eines TCP-Moduls. Die *kursiv* geschriebenen Übergangsbedingungen sind die entsprechenden *service requests* einer Applikation, die TCP verwendet. Sowohl Client- als auch Serverapplikationen starten im *CLOSED* Zustand. Eine Server-Applikation (z.B. ein Telnet-Server) fordert vom TCP-Modul ein *Passive Open* mit der entsprechenden Source-Port-Nummer (in diesem Fall well known Port 23) an. Der Destination-Port (bzw. Socket) bleibt unspezifiziert. Die Server-Applikation geht in den *PASSIVE_OPENED* Zustand über und wartet auf eine Anforderung des Clients. Eine Client-Applikation fordert vom TCP-Modul ein *Active Open* an und

spezifiziert dabei den Destination Socket. Die Source-Port-Nummer ist der nächste frei im System verfügbare (local port). Der Client landet im fehlerfreien Fall im *ACTIVE_OPENED* Zustand. Zwischen dem TCP-Modul des Servers und dem des Clients findet nun der Verbindungsaufbau, der *3-Wege-Handshake*, statt.

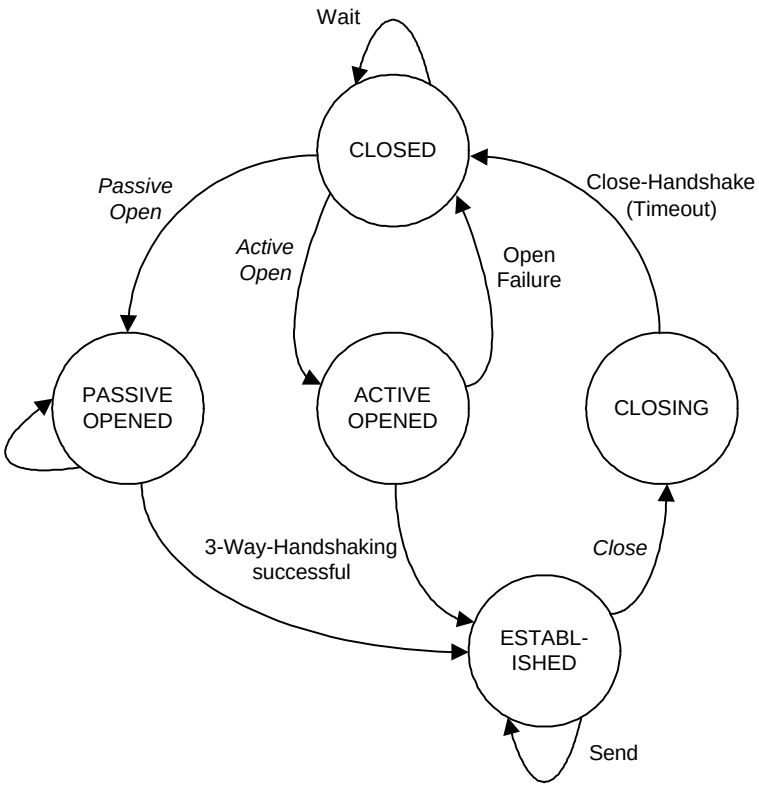


Abbildung 3.1: TCP-Zustandsmaschine

Nach erfolgreichem Verbindungsaufbau befinden sich sowohl der Server als auch der Client im Zustand *ESTABLISHED*. Nun können Daten (vollduplex, d.h. in beide Richtungen) ausgetauscht werden. Spezielle Mechanismen zur Kontrolle der Segmentnummern der TCP-Segmente stellen sicher, dass die Daten vollständig, sicher und in der richtigen Reihenfolge auf der anderen Seite ankommen und bestätigt werden.

Entweder der Server oder der Client fordert einen Verbindungsabbau an (*close request*). Im Zustand *CLOSING* sind alle notwendigen Aktivitäten vereint, um sowohl den Client als auch den Server in den Ausgangszustand *CLOSED* zu bringen.

Für das Analyseprogramm sind diese Zustands- und Übergangsinformationen allerdings nicht verwertbar. Das Messsystem mit der DAG-Karte und *recordsneaker* haben in die Funktionen der beteiligten Kommunikationspartner keinen Einblick. *recordsneaker* kann nur Schlussfolgerungen aus dem aktuellen Verlauf der TCP-Kommunikation anhand der Pakete, die es aufnimmt, ziehen. Dazu sind die folgenden beiden Schritte notwendig:

1.) Anhand des in [1] verwendeten Flow-Modells sind die zu einer TCP-Verbindung gehörenden Pakete erkennen.

2.) Anhand von Informationen aus den Paketen (Header) sind Schlussfolgerungen über den Verlauf der TCP-Verbindung ziehen.

Diese Schlussfolgerungen sind jedoch nur für TCP-Verbindungen möglich, die der in [1] genannten Beschränkungen des Modells unterliegen. Das heißt, dass in diesem Fall nur TCP-Verbindungen untersucht werden können, die mit einem festen Source- und Destination Socket arbeiten und bei denen die Kommunikationspartner auf dem gleichen Socket senden, auf dem sie eine Antwort erhalten haben.

Für die Verbindungssteuerung sind im TCP-Segment im Header 3 Bits von Bedeutung

- **SYN** (Synchronisationsbit, Anfrage eines definierten Verbindungsstarts)

- **FIN** (Ende-Bit, Anfrage eines definierten Verbindungsendes)

- **ACK** (Bestätigungs-Bit, Bestätigung eines Service Requests oder eines Datensegments)

Zusätzlich werden für die Verbindungssteuerung **Sequenznummern** ausgetauscht. Diese sind auch während der eigentlichen Nutzdatenübertragung von Bedeutung. Dort ermöglicht der PAR-Mechanismus (*positive acknowledge with retransmission*) und die elementare Flusskontrolle (*sliding window mechanism*) mit Hilfe der Sequenznummern eine reibungslose Kommunikation. Sie spielen aber auch eine entscheidende Rolle für die Beurteilung der Verbindungssteuerung. Abbildung 3.2 zeigt die im TCP-Header relevanten Informationen, die für die Verbindungssteuerung von Bedeutung sind. Eine vollständige Beschreibung des TCP-Headers ist in [2] zu finden.

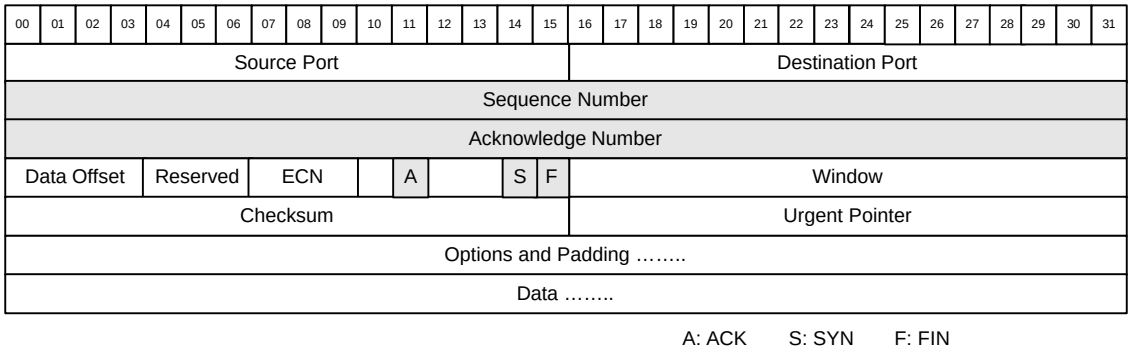


Abbildung 3.2: TCP-Header

Das Übergangsdiagramm der TCP-Verbindung in der so genannten *Mealy-Notation* stellt einen Blick von außen auf TCP-Module der miteinander kommunizierenden Stationen dar. Anhand der Zustandsübergänge muss die Messeinrichtung nachvollziehen können, in welchem Zustand sich die TCP-Module der beiden Kommunikationspartner gerade befinden. Vorher auftretende Übergänge müssen gespeichert und auf aktuelle Übergänge muss reagiert werden, ebenso, wie es die Verbindungsstacks der beiden Partner tun. Damit können dann Aussagen bezüglich des Verlaufs der Kommunikation getroffen werden.

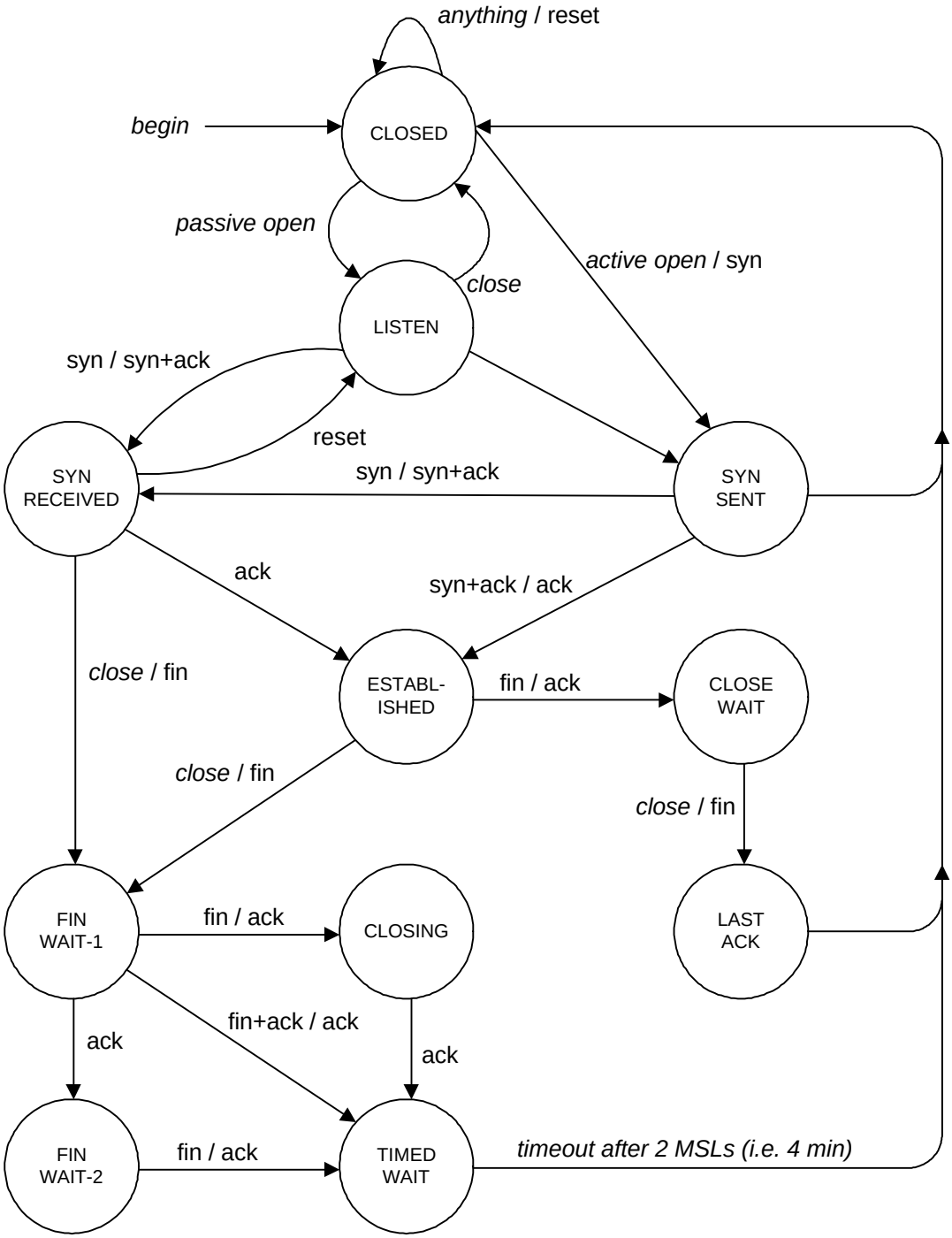


Abbildung 3.3: TCP Mealy-Notation

Da für die Erweiterung von *recordsneaker* das erste Paket nach einem erfolgreichen TCP-Verbindungsaufbau zu untersuchen ist, wird dieser noch

etwas näher betrachtet. *recordsneaker* muss für eine sichere Erkennung eines erfolgreichen Verbindungsaufbaus exakt diesen Ablauf nachvollziehen. Hierbei steht das schon erwähnte 3-Wege-Handshaking im Vordergrund, das sich aus der Abbildung 3.4 ergibt.

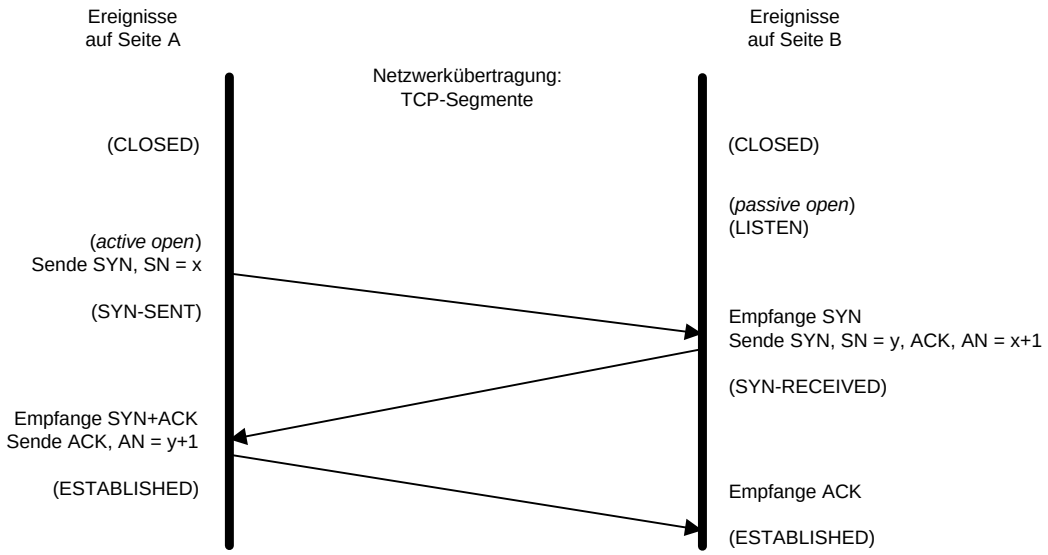


Abbildung 3.4: TCP Verbindungsaufbau

1. Station A will eine TCP-Verbindung zu Station B aufbauen und sendet dazu ein TCP-Segment mit gesetztem (aktivem) SYN-Flag und der Initial Sequence Number (ISN) x im Sequence Number (SN) Feld des TCP-Headers.

2. Station B empfängt diesen SYN-Request und antwortet ihrerseits mit der Bestätigung der ISN im Acknowledge Number (AN) Feld durch den Wert x+1 und gesetztem ACK-Flag. (Die Sequenznummer im AN-Feld ist so zu interpretieren, dass es die für das nächste von der Gegenstelle gesendete Segment erwartete Sequenznummer charakterisiert.) Im selben Segment übermittelt Station B and die Station A ihre ISN y im Sequence Number (SN) Feld und setzt das SYN-Flag.

3. Station A empfängt ein Segment mit gesetztem ACK und SYN. ACK bedeutet hier, dass Station B den Verbindungswunsch inklusive ISN akzeptiert. SYN zeigt an, dass Station B im Sequence Number Feld ihre ISN übermittelt. Station A quittiert dies mit einem ACK Segment; im Acknowledge Number Feld steht y+1 als Bestätigung der ISN von Station B. Nachdem Station B dieses ACK empfangen hat, ist die virtuelle, vollduplexe Verbindung zwischen den Stationen A und B aufgebaut. Die Datenoktettnummerierung von A nach B beginnt mit x+1 und die von B nach A mit y+1.

An dieser Stelle muss von *recordsneaker* sowohl ein von A nach B als auch in Gegenrichtung kommendes Paket untersucht werden. Von beiden kann nun ein Nutzdaten-Paket erwartet werden. Gleichzeitig muss immer wieder kontrolliert werden, ob nicht neue SYN-Anforderungen auftreten, denn das bedeutet gleichzeitig das Ende einer vorherigen TCP-Verbindung mit den festgelegten Flow-Eigenschaften. Zwar findet in einer vollständigen TCP-Kommunikation auch einen ordnungsgemäßer Verbindungsabbau (FIN Anforderung und seine Folgesegmente) statt, jedoch können Verbindungsabbrüche, Timeouts oder andere Ereignisse dazu führen, dass die Messeinrichtung einen solchen Ablauf nicht registriert. Durch das 3-Wege-Handshaking zu Beginn ist aber ein neuer TCP-Verbindungsaufbau immer eindeutig charakterisiert. Daher konzentriert sich die in Kapitel 5 beschriebene Umsetzung in *recordsneaker* ausschließlich auf die Überwachung dieses Teils der TCP-Verbindung.

4 Schnittstelle zur MySQL-Datenbank

Zunächst einmal muss eine MySQL-Datenbankinstallation durchgeführt werden. Diese kann auf dem System laufen, auf dem *recordsneaker* arbeitet. Im Echtzeitbetrieb ist jedoch die Installation auf einem Fremdsystem, das über eine Netzwerkschnittstelle verfügbar ist, zu empfehlen. So stehen sowohl für *recordsneaker* als auch für den MySQL-Server die jeweils maximalen Systemressourcen zur Verfügung. An die MySQL-Installation selbst sind keine hohen Anforderungen zu stellen. Diese hängen nur davon ab, welche Funktionalitäten für die Auswertung der Flowdaten in der Datenbank benötigt werden. Die Kommunikation zwischen *recordsneaker* und dem MySQL-Server wird mit Hilfe einer von MySQL zur Verfügung gestellten API realisiert. Die MySQL-API weist folgende Leistungsmerkmale auf:

- Routinen zur Verbindungsverwaltung, die eine Sitzung mit dem Server einrichten oder beenden

- Routinen, mit denen Anfragen konstruiert und an den Server geschickt werden und mit denen die Ergebnisse verarbeitet werden

- Funktionen zur Meldung von Fehlern und Statuszuständen, um die genaue Ursache für das Fehlschlagen anderer API-Aufrufe ermitteln zu können

- Routinen, mit denen Optionen verarbeitet werden können, die über die Optionsdateien oder die Befehlszeile übergeben wurden

Für die Kommunikation zwischen *recordsneaker* und dem MySQL-Server werden aus der MySQL-API nur wenige Kommandos genutzt. Die Schnittstelle zur MySQL-Datenbank stellt sich für *recordsneaker* als ein zusätzliches Ausgabemodul da, die es in dieser Version neben den beiden in [1] implementierten anbietet.

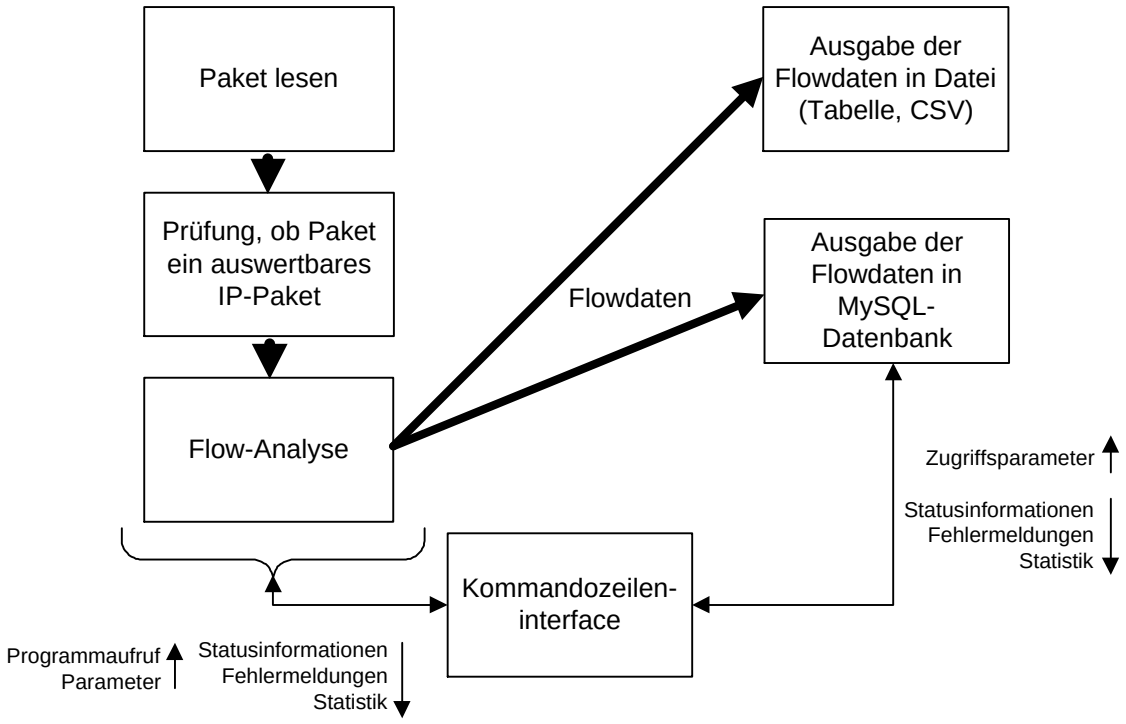


Abbildung 4.1: Erweitertes recordsneaker Programmschema

Das *recordsneaker*-Ausgabemodul für MySQL liefert die Flowdaten an die MySQL-Netzwerkschnittstelle. Ein Auslesen von Datenbankinformationen durch *recordsneaker* ist nicht erforderlich, lediglich Status- und Fehlerinformationen sollten zurück gegeben werden, um den Nutzer über den Verlauf der Datenübertragung an MySQL und eventuelle Fehler zu informieren.

Eine vollständige Beschreibung der MySQL C-API findet sich in [4]. Für die Entwicklung von MySQL-Clients in C gibt es viele Dokumentationen, ein sehr ausführliches und reichhaltiges Tutorial ist in [5] zu finden.

5 Programmtechnische Umsetzung

Neben den beiden Hauptaufgaben, der MySQL-Datenbankintegration und der Analyse der TCP-Payload wurden gegenüber der in [1] entwickelten Variante von *recordsneaker* noch einige kleine Dinge verändert, auf die an dieser Stelle nur kurz eingegangen werden soll:

- Erhöhung der Programmgeschwindigkeit durch Entfernen einiger Zwischenspeicherungen und Reduzierung von Kopiervorgängen im physikalischen Speicher

- Ausgabe der IP-Adressen (in der CSV-Dateiausgabe und der folgenden MySQL-Datenbankausgabe) in integer-Form zur besseren Verarbeitung

- Auswertung der Interface ID der DAG-Messkarte als Indikator für die Sende- und Empfangsrichtung

Die Erkennung der ersten Payload eines TCP-Segments nach erfolgreichem Handshaking

Die zusätzlich zu den von *recordsneaker* in [1] gelieferten Informationen müssen beim Auslesen des jeweiligen Paketes als Ergänzung zu den schon durchgeführten ausgelesen werden. Die weitere Analyse findet in einem zusätzlichen Zweig, in den bei einem erkannten TCP-Segment verzweigt wird,

statt. Die gesamte eigentliche Flowanalyse bleibt davon unberührt. Um die Verfolgung des 3-Wege-Handshakings von TCP, welches in Kapitel 3 näher erläutert wurde, in *recordsneaker* zu integrieren, sind mehrere Komponenten erforderlich:

- das direkte Auslesen und Zwischenspeichern von **SYN** und **ACK** Flag sowie der Sequenznummern aus dem **SN** (Sequence Number) und **AN** (Acknowledgement Number) Feld eines erkannten TCP-Segments

- eine Analyse, welche Schritte des Handshakings der Flow abgearbeitet hat und ggf. ob das Handshaking nicht unterbrochen wurde

- bei erfolgreichem Handshaking Auslesen der Payload des nächsten zum Flow zugehörigen TCP-Segmentes

Dem Flow-Element wurden folgende Datenfelder hinzugefügt:

tcp_seq_a	Die ISN (initial sequence number) für Station A
tcp_seq_b	Die ISN (initial sequence number) für Station B
tcp_syn_up	SYN-Flag in Up-Richtung schon gesehen?
tcp_syn_down	SYN-Flag in Down-Richtung schon gesehen?
tcp_hs_ok	Handshaking erfolgreich verlaufen?

Über die Problematik der Definition von Up- und Down-Richtung und welche Station A und welche B ist, wurde in [1] schon diskutiert. Die hier verwendete Definition für A und B und Up und Down ist aber prinzipiell unabhängig von der in der eigentlichen Flow-Analyse verwendeten. Die zu analysierenden Zustände im Rahmen der Handshaking-Verfolgung haben keinen Einfluss auf die verwendete Flow-Definition und deren Umsetzung.

Folgender Ablaufplan ist in den TCP-Zweig von **flowfunc.c** integriert worden:

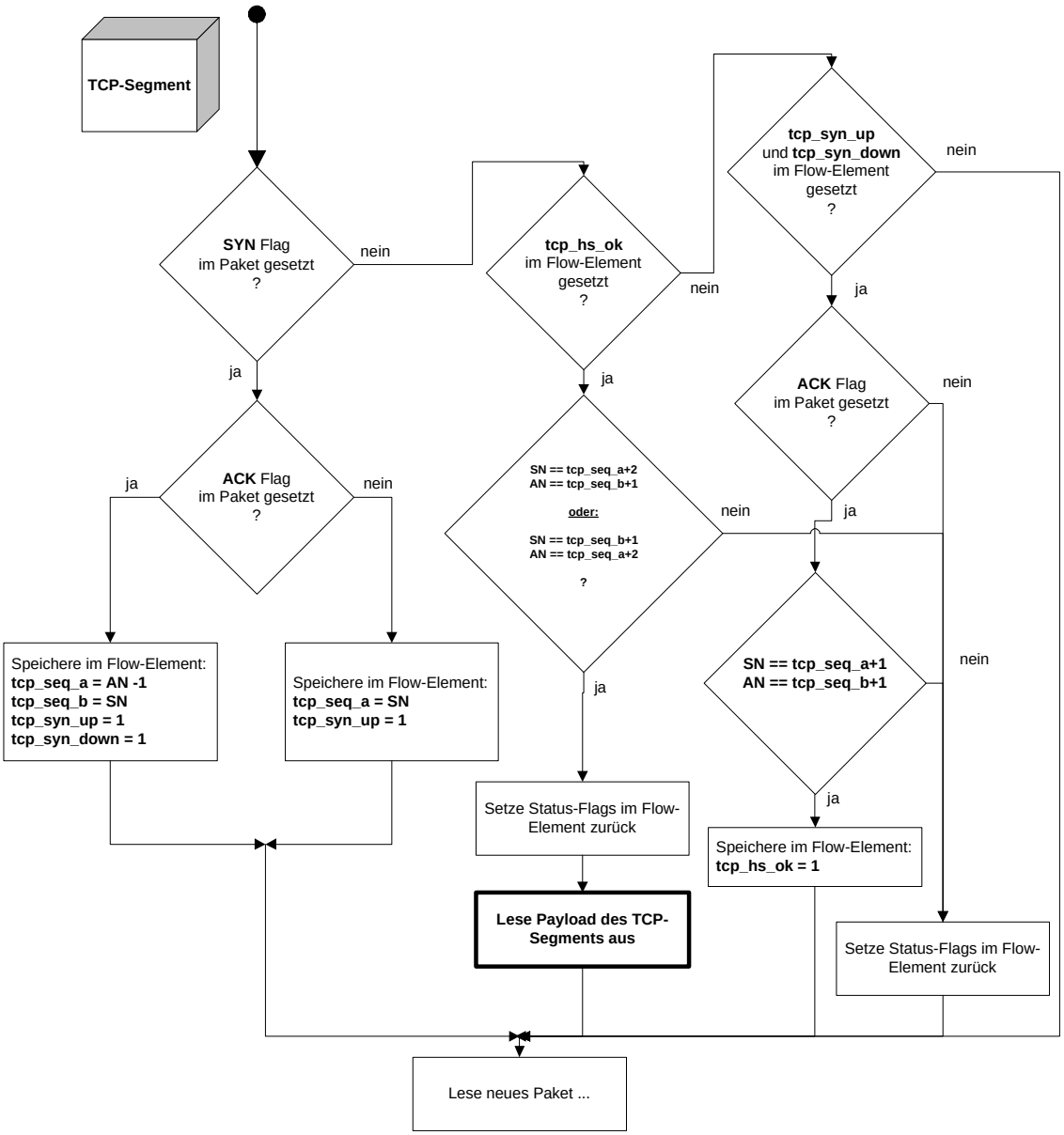


Abbildung 5.1: Erkennung des TCP-Handshakings

Dieser Ablauf entspricht weitestgehend dem in Kapitel 3 erklärten Zustandsdiagramm. Für die Handshaking-Analyse sind also prinzipiell die gleichen Schritte erforderlich wie in dem entsprechenden TCP-Modul der kommunizierenden Stationen. Hier müssen also Flags überwacht und mit

schon „gesehenen“ verglichen werden, aus absoluten Sequenznummern müssen relative berechnet werden usw. Erst, nachdem das Handshaking erfolgreich war – das *tcp_hs_ok* Flag im Flow-Element ist gesetzt - wird die Payload des nächsten TCP-Segments gespeichert. Das passiert jedoch nur unter der Voraussetzung, dass die entsprechenden Segmentnummern übereinstimmen, ansonsten wird alles verworfen und die Analyse der TCP-Payload gilt für diese Verbindung als nicht erfolgreich.

Einen zusätzlichen Vorteil hat diese Analyse jedoch gegenüber einem „echten“ TCP-Modul: Sollte der Anfang des Handshakings, also das erste *SYN*, vom Messsystem bzw. von *recordsneaker* übersehen worden sein, so funktioniert die Erkennung des Verbindungsaufbaus dennoch, da anhand des folgenden Paketes von der Gegenstelle mit *SYN* und *ACK* von einem ursprünglichen Senden des *SYN* ausgegangen werden kann. Die für die Analyse erforderliche ISN (in *tcp_seq_a* zu speichern) lässt sich aus der *AN* des zweiten Pakets berechnen und speichern. In einer richtigen TCP-Kommunikation zwischen 2 Stationen würde laut TCP-Zustandsdiagramm mit einem *SYN/ACK* als erstem Paket dieses verworfen und die Verbindung zurückgesetzt werden.

Speicherung der Flowdaten in einer MySQL-Datenbank

Die Ausgabemöglichkeit in eine MySQL-Datenbank wurde separates Ausgabemodul in *recordsneaker* implementiert. Zusätzlich zu den in [1] gegebenen Ausgabemöglichkeiten Tabelle und CSV-Datei ist die Ausgabe der Flowdaten in ein vordefiniertes MySQL-Datenbankschema möglich. Dieses Datenbankschema enthält die gleichen Einträge wie die CSV-Datei, zusätzlich noch einen Eintrag für die TCP-Payload. Das komplette Datenbankschema ist in Anhang A zu finden.

Für die Aufwertung von *recordsneaker* als MySQL Client sind folgende Includes aus dem Entwicklerpaket von MySQL nötig: **my_alloc.h**, **mysql_com.h** und **mysql_version.h**. Ferner muss die **mysqlclient.a** Bibliothek gelinkt werden.

Zunächst wird mit **mysql_init()** die Verbindung zum MySQL-Server initiiert und ein Verbindungs-Handle zurückgegeben, der Informationen über die Verbindung enthält. Mit **mysql_real_connect()** wird die Verbindung zum Server hergestellt. Dabei müssen folgende Parameter übergeben werden:

- **Zeiger auf den Verbindungshandle**; der von *mysql_init()* zurückgegebene Wert

- **Serverhost**; Hostname oder IP-Adresse des MySQL-Servers; default-Wert ist localhost, bei NULL wird ebenfalls localhost gesetzt

- **Benutzername und Kennwort**; Ist der Name NULL, dann sendet die Client-Bibliothek den aktuellen Anmeldenamen an den Server, ist das Kennwort NULL, so wird kein Kennwort an den Server gesendet

- **Name der Datenbank**, die nach Herstellung der Verbindung als Standard-Datenbank gewählt wird; ist der Wert NULL, so wird keine Datenbank ausgewählt

- **Portnummer und Socket-Datei**; Die Portnummer wird für TCP/IP-Verbindungen zum Server benötigt, der Socket-Name für Socket-Verbindungen benutzt; die Werte 0 und NULL weisen die Client-Bibliothek an, die Standardwerte zu verwenden

- **Flags-Wert**; für spezielle Verbindungsoptionen

Im Programm *recordsneaker* selbst sind folgende default-Werte gesetzt, die übergeben werden, wenn der entsprechende Parameter in der Kommandozeile nicht angegeben wird:

Hostname: localhost

Benutzername: mysql

Passwort: kein Passwort

Die Angabe der Namen von Datenbank und Tabelle sind obligatorisch. Mit **mysql_query()** wird dann die eigentliche Datenübertragung von *recordsneaker* in die entsprechenden Tabellenfelder durchgeführt. Die Notation erfolgt in SQL-Standardsyntax.

Da *recordsneaker* nicht primär dazu ausgelegt ist, ein vollwertiger MySQL-Client zu sein, sondern um hauptsächlich einfach, unkompliziert und vor allem schnell die Flowdaten in eine MySQL-Datenbank zu schreiben, wurde auf das vollständige Einräumen aller Möglichkeiten bei der Kommunikation mit dem MySQL-Server verzichtet. Das Datenbankschema muss beispielsweise vorher im MySQL-Server angelegt werden und der Datenbankname, der Tabellenname und die Datenfelder müssen stimmen. Jedoch ist im Fehlerfall gewährleistet, dass auf der Standardausgabe (Shell) aussagekräftige Meldungen ausgegeben werden und die Programmabarbeitung definiert beendet wird. Mit Hilfe der MySQL-C-API Aufrufe **mysql_error()** und **mysql_errno()** werden vom Server entsprechende Informationen an *recordsneaker* zurückgegeben. Mit **mysql_close()** wird nach Beendigung der Übertragung die Datenverbindung zum MySQL-Server ordnungsgemäß abgeschlossen.

Die Eingabemöglichkeiten auf der Kommandozeile erweitern sich mit der zusätzlichen Ausgabemöglichkeit um folgende Parameter:

-d --databasename=DATABASENAME

MySQL Datenbank, zu der verbunden werden soll

-D --db_mysql

Ausgabe in eine MySQL Datenbank erzeugen

-h --hostname=HOSTNAME

MySQL Server zu dem verbunden werden soll, Standard ist "localhost"

-n --nofile

Keine (zusätzliche) Dateiausgabe erzeugen

-p --password=PASSWORD

Passwort für den MySQL-Benutzer

-T --tablename=TABLENAME

Name der Tabelle in der MySQL Datenbank, in die die Flowdaten geschrieben werden sollen

-u --username=USERNAME

Benutzername für die MySQL Serververbindung, Standard ist "mysql"

6 Ergebnisse und Ausblick

Der Test auf die Funktionalität von *recordsneaker* in Hinblick auf die Erzeugung der Flowdaten ist in [1] erfolgt. Im Rahmen dieser Arbeit war in erster Linie zu prüfen, ob eine ordnungsgemäße Ausgabe der Daten in eine MySQL-Datenbank erfolgt und ob das Auslesen der Payload eines TCP-Segmentes nach erfolgreichem Handshaking korrekt funktioniert.

Der Funktionstest wurde mit einer lokalen MySQL-Serverinstallation durchgeführt. Der MySQL-Server lag dabei in der Version 4.0.23 vor. Es ist davon auszugehen, dass die Kommunikation mit allen MySQL-Distributionen ab Version 4.x gelingt, da keine der Erweiterungen in diesen Paketen verwendet wurden. Als Database Engine wurde *InnoDB* verwendet. Das spielt jedoch im Zusammenspiel mit *recordsneaker* keine Rolle, sondern beeinflusst die Performance bei der nachfolgenden Auswertung der Flowdaten in der Datenbank. Ansonsten wurde der Server in sämtlichen Standardeinstellungen belassen.

Im MySQL-Server wurde eine Datenbank mit dem Namen *dagdata* angelegt und darin eine Tabelle mit dem Namen *flowdata* erzeugt. Diese Einstellungen wurden dann in der Kommunikation zwischen *recordsneaker* und dem MySQL-Server als Kommandozeilenparameter verwendet. Die Tabelle wurde mit Hilfe des MySQL Administrator in der Version 1.0.16 von einem Windows PC aus generiert. Dieses Programm gibt sich gegenüber dem MySQL-Server als MySQL-Client in der Version 5.0.0 aus und ermöglicht die Datenbank- und Tabellenkonfiguration mit Hilfe einer grafischen Oberfläche. Damit konnte

gleichzeitig auf einfache Weise der Verlauf und Erfolg der Datenübertragung von *recordsneaker* in die Datenbank überwacht werden. Das Datenbankschema lässt sich auch direkt in der Kommandozeile des MySQL-Servers einbinden. Das vollständige Datenbankschema und die SQL-Syntax zur Einbindung per Kommandozeile sind in Anhang A zu finden.

Die Kommunikation zwischen *recordsneaker* und dem MySQL-Server funktioniert reibungslos. Für die Tests wurden die selben wie in [1] verwendeten Beispiel-Tracefiles eingesetzt. Bei der Ausgabe über das MySQL-Modul war festzustellen, dass die Programmabarbeitung etwas langsamer erfolgt als bei der Ausgabe in eine Datei. Um jedoch auszutesten, ob sich diese Verlangsamung bei größeren Datenmengen als kritisch erweisen könnte, müsste man Tests mit Echtzeitdaten durchführen. Bei den genutzten Tracefiles stößt man an keine Grenzen des physikalischen Speichers des Rechnersystems. Ein Großteil der Übertragung zum MySQL-Server fand daher statt, nachdem *recordsneaker* die Flowdaten schon erzeugt hatte und keine neuen Eingangsdaten mehr vorlagen. Es ist aber davon auszugehen, dass es zu keinen Problemen bei der Ausgabe kommt. Anhang C zeigt den Auszug aus einer mit *recordsneaker* erzeugten Tabelle der MySQL-Datenbank.

Für die Erkennung der Payload eines TCP-Flows war zu ermitteln, ob *recordsneaker* genau dann die Paketinformationen eines TCP-Segmentes ausliest, wenn zuvor ein erfolgreiches 3-Wege-Handshaking stattgefunden hat, wie es in Kapitel 3 beschrieben ist. Das Ergebnis wurde mit Hilfe von *Etherreal* in der Version 0.10.7 verifiziert ([6]). Mit Hilfe der Filterfunktionen lassen sich in Etherreal einzelne TCP-Verbindungen verfolgen. Hierzu wurden die Filter auf beispielhaft ausgewählte TCP-Sockets eingestellt. In der Paketverfolgung ließ sich dann der gesamte Verbindungsverlauf der Socketpaare nachvollziehen. So war ein entsprechend verlaufenes TCP-Handshaking ebenso sichtbar wie die von der DAG-Messkarte (wenn auch in der Regel nur unvollständig) mitgeschnittene Higher Layer Payload. War eine TCP-Verbindung erfolgreich aufgebaut worden, ließ sich die Payload des nächsten Segmentes direkt

ablesen, bei einigen bekannten Protokollen benannte Etherreal noch weitere Informationen bezüglich der Anwendungsdaten. Die abgelesenen Payload-Daten waren dann mit den von *recordsneaker* erzeugten Daten zu vergleichen. Dies geschah in der Ausgabe der MySQL-Datenbank, da *recordsneaker* die Payload-Daten nur dort hineinschreibt.

Die Untersuchung auf die richtige Erkennung von TCP-Payloads ergab folgende aufgetretenen Fälle:

- Eine TCP-Payload des relevanten TCP-Segments eines Flows wurde nicht erkannt, da innerhalb des Messzeitraumes entweder kein erfolgreiches TCP-Handshaking stattfand oder von der Messeinrichtung keines erfasst wurde.

- Eine TCP-Payload des relevanten TCP-Segments eines Flows wurde nach einem erfolgreichen vollständigen TCP-Handshaking erkannt und gespeichert.

- Eine TCP-Payload des relevanten TCP-Segments eines Flows wurde erkannt und gespeichert, wenn das erste SYN-Segment der Kommunikation von der Messeinrichtung nicht erfasst wurde.

Im Anhang C finden sich drei Beispiele aus den Messdaten, in denen die o.g. Fälle aufgetreten sind. Dort sind die entsprechenden TCP-Stream in einem Ethereal Screenshot und das von *recordsneaker* ermittelte Ergebnis dargestellt.

Häufig sichtbare Payloads waren TCP-Segmente als Anfragen an Webserver (häufig verwendeter Port 80), in denen der HTTP-Request „*GET /<Verzeichnis>*“ auftrat. Die Kommunikation zwischen einem Webbrowser und einem Webserver ist in der Regel nur wenige Sekunden lang, daher waren solche TCP-Verbindungen in den Messdaten häufig zu beobachten. Sollte es innerhalb des Messzeitraumes jedoch mehrere Anfragen ein und desselben

Sockets (IP-Adresse und Layer 4 Port identisch) an den genannten Server geben, so speichert *recordsneaker* nur die Payload nach dem letzten Handshaking. Das wurde aus zwei Gründen so gelöst:

1.) Das Bereithalten von Speicher für eine nicht vorhersehbare Anzahl von Payloads eines TCP-Flows ist nicht praktikabel, da sich diese von 0 bis zu mehreren tausend bewegen kann. Die Speicherausnutzung sowohl in *recordsneaker* als auch in der MySQL-Datenbank sollte jedoch möglichst effizient sein.

2.) Es ist davon auszugehen, dass die eventuell mehrfach gespeicherten Payloads einer TCP-Verbindung nach dem verwendeten Flow-Modell große Ähnlichkeit aufweisen und mitunter gar identisch sind. So wird eine Anfrage an einen Webserver (identisches Socketpaar) im Allgemeinen immer wieder die Struktur „*GET /<Verzeichnis>*“ haben. Daher erscheint eine mehrfache Speicherung wenig sinnvoll.

Es gab jedoch auch viele Verbindungen, die im Messzeitraum überhaupt keine *SYN*-Bits aufweisen. Die Analyse der Segmentnummern in Etherreal lässt in diesen Fällen darauf schließen, dass diese der Messeinrichtung nicht etwa „entgangen“ sind, sondern dass im Messzeitraum kein Versuch stattgefunden hat. Besonders charakteristisch für solche TCP-Verbindungen sind Übertragungen von Peer-to-Peer-Anwendungen, die sich über mehrere Stunden hinziehen können.

Sollte die in 1.) genannte Beschränkung ein Hindernis bei der späteren Auswertung darstellen, so ließe sich das verwendete Flow-Modell dahingehend erweitern, dass ein eingeleitetes TCP-Handshaking bei der Verbindung eines bestimmten Socketpaares gleichzeitig als Beendigung eines vorhergehenden Flows auf ein und demselben Socketpaar definiert wird. Dies entspräche auch der Logik einer Verfeinerung des Verkehrsflussmodells hin zur anwendungsbezogenen Betrachtung. Es bleibt jedoch die schon in [1]

beschriebene Begrenzung des Konzeptes von *recordsneaker*, dass Anwendungen, die mehrere und eventuell wechselnde Layer 4 Ports bei der Übertragung verwenden, mit dem hier verwendeten Verkehrsmodell nicht erfasst werden. Eine Lösung bietet nur die Erweiterung des Modells unter Zuhilfenahme von weiteren Informationen über die kommunizierenden Anwendungen (verwendete Ports, bestimmte Reihenfolge im Kommunikationsablauf etc.). Denkbar wäre auch eine Verlagerung dieser Problematik in die Auswertung der Flowdaten in der MySQL-Datenbank. Eine weitestgehende Trennung zwischen Erfassung der Flowdaten mittels *recordsneaker* nach dem gegebenen Flow-Modell und der Erweiterung desselben und der Interpretation aus den in der MySQL-Datenbank gespeicherten Daten käme der Geschwindigkeit der Datenerfassung entgegen. Gleichzeitig stünde diese Zweistufigkeit einer Echtzeitanalyse im Weg.

7 Literaturverzeichnis

[1] Christian Manthey, ***Entwicklung einer Softwareanwendung für die floworientierte Auswertung von IP Datenverkehr***, Kleiner Beleg, Universität Rostock, *2004*

[2] Jon Postel, ***RFC 793, Transmission Control Protocol***, Einreichung beim Information Sciences Institute, University of California, September 1981

[3] MySQL AB, **MySQL**, Dokumentation, http://www.mysql.com/, Letzte Änderung: 10. März 2005, Letzter Besuch: 11. März 2005

[4] MySQL AB, **MySQL C API Documentation**, Beschreibung und Tutorial, http://dev.mysql.com/doc/mysql/en/c.html, Letzte Änderung: 18. Dezember 2004, Letzter Besuch: 11. März 2005

[5] Paul DuBois, **The definitive guide to using, programming and administering MySQL 4**, Sams Developer´s Library, 2. Aufl. 2003

[6] Verschiedene Entwickler, **Ethereal**, Programmbeschreibung, http://www.ethereal.com, Letzte Änderung: 26. Mai 2004, Letzter Besuch: 23. Juni 2004

[7] Guido Krüger**, Go To C-Programmierung**, Addison-Wesley, 4. Aufl. 1998

[8] Helmut Herold, Jörg Arndt, **C-Programmierung unter Linux**, SuSE PRESS, 1. Aufl. 2001

Anhang A: Verwendetes MySQL-Datenbankschema

Field	Type	NULL	Key	Default	Extra
Flow_ID	int(10) unsigned		PRI	NULL	auto_inc
Interface_ID	int(10) unsigned	YES		0	
Source_IP	int(10) unsigned	YES		0	
Destination_IP	int(10) unsigned	YES		0	
Source_Port	int(10) unsigned	YES		0	
Destination_Port	int(10) unsigned	YES		0	
L4_Encapsulation_Type	int(10) unsigned	YES		0	
Flow_Activity_Time	int(10) unsigned	YES		0	
Packets_Up	int(10) unsigned	YES		0	
Packets_Down	int(10) unsigned	YES		0	
Packets_Sum	int(10) unsigned	YES		0	
Bytes_Up	int(10) unsigned	YES		0	
Bytes_Down	int(10) unsigned	YES		0	
Bytes_Sum	int(10) unsigned	YES		0	
ICMP_Type	int(10) unsigned	YES		0	
Flow_End_Reason	int(10) unsigned	YES		0	
TCP_Payload	varchar(46)	YES		0	

SQL-Query zum Generieren des MySQL-Datenbankschemas:

```
CREATE TABLE `dagdata`.`neu` (
  `Flow_ID` INTEGER UNSIGNED AUTO_INCREMENT,
  `Interface_ID` INTEGER UNSIGNED,
  `Source_IP` INTEGER UNSIGNED,
  `Destination_IP` INTEGER UNSIGNED,
  `Source_Port` INTEGER UNSIGNED,
  `Destination_Port` INTEGER UNSIGNED,
  `L4_Encapsulation_Type` INTEGER UNSIGNED,
  `Flow_Activity_Time` INTEGER UNSIGNED,
  `Packets_Up` INTEGER UNSIGNED,
  `Packets_Down` INTEGER UNSIGNED,
  `Packets_Sum` INTEGER UNSIGNED,
  `Bytes_Up` INTEGER UNSIGNED,
  `Bytes_Down` INTEGER UNSIGNED,
  `Bytes_Sum` INTEGER UNSIGNED,
  `ICMP_Type` INTEGER UNSIGNED,
  `Flow_End_Reason` INTEGER UNSIGNED,
  `TCP_Paylad` VARCHAR(46),
  PRIMARY KEY(`Flow_ID`)
)
TYPE = InnoDB;
```

Anhang B: Quellcodebaum

/

/include
- buffer.h
- config.h
- dag_analysis.h
- dag_frame.h
- dag_rawio.h
- dag_recordinfo.h
- dag_types.h
- flowfunc.h
- icmp_header.h
- ip_header.h
- mysql.h
- mysql_com.h
- mysql_version.h
- my_alloc.h
- rec_io.h
- tcp_header.h
- udp_header.h

/lib
- libmysqlclient.a
- librawio.a
- librecio.a

/progs
- buffer.c
- dag_analysis.c
- flowfunc.c
- readrecord.c
- recordsneaker.c

/raw_drivers
- dag_drv.c
- dag_drv.h
- dag_rawio.c
- file_drv.c
- file_drv.h

/rec_drivers

dag_recdrv.c
dag_recdrv.h
pcap_recdrv.c
pcap_recdrv.h
rec_io.c

MYSQL_SCHEME
README

Anhang C: Beispielausgaben

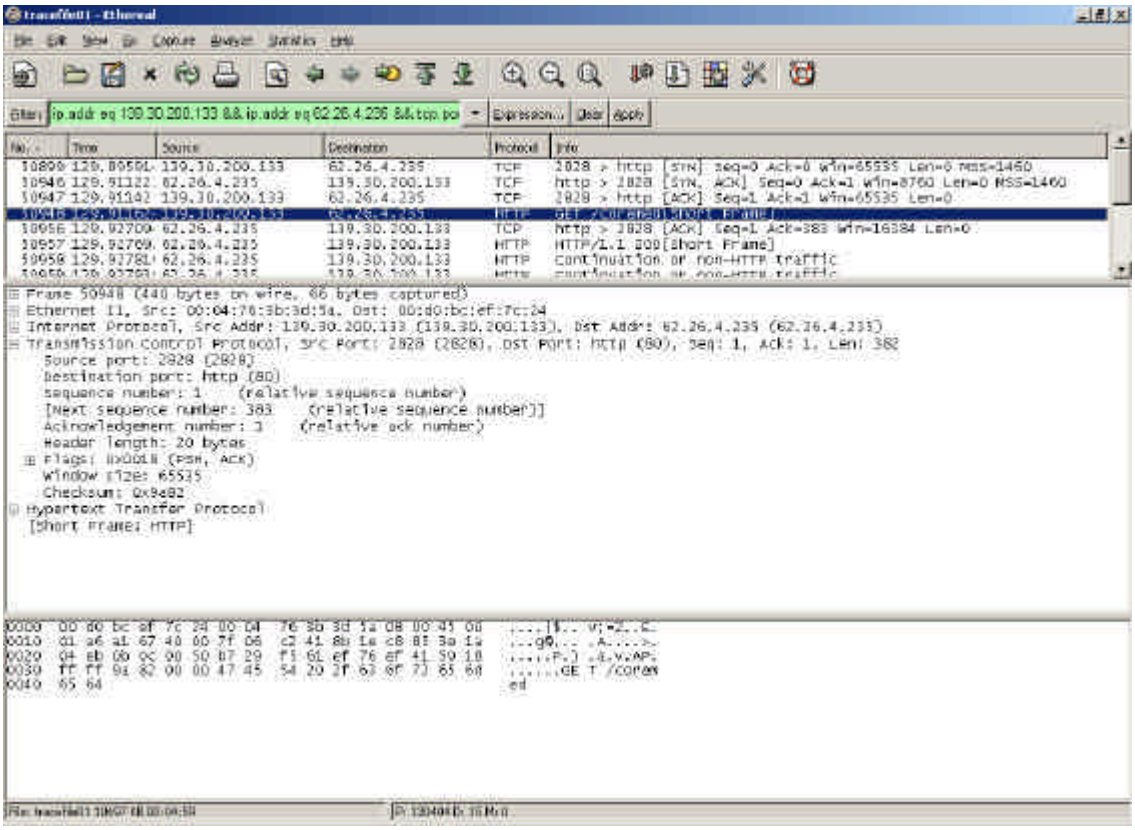
Recordsneaker – Auszug aus einer MySQL-Beispielausgabe

```
+---------+-------------+------------+---------------+------------+-----------------
+----------------------+--------------------+------------+-------------+------------
+----------+------------+-----------+-----------+----------------+-------------+
| Flow_ID | Interface_ID | Source_IP  | Destination_IP | Source_Port | Destination_Port
| L4_Encapsulation_Type | Flow_Activity_Time | Packets_Up | Packets_Down | Packets_Sum |
Bytes_Up | Bytes_Down | Bytes_Sum | ICMP_Type | Flow_End_Reason | TCP_Payload  |
+---------+-------------+------------+---------------+------------+-----------------
+----------------------+--------------------+------------+-------------+------------
+----------+------------+-----------+-----------+----------------+-------------+
|    6711 |            1 | 2334050528 |     2334050559 |         138 |              138
|                   17 |                  0 |          2 |           0 |           2 |
510 |          0 |       510 |         0 |               1 | 0            |
|    6712 |            1 | 2334050526 |     2334050559 |         138 |              138
|                   17 |                  0 |          2 |           0 |           2 |
510 |          0 |       510 |         0 |               1 | 0            |
|    6713 |            1 | 2334052354 |     2334050403 |         138 |              138
|                   17 |                 21 |          4 |           0 |           4 |
909 |          0 |       909 |         0 |               1 | 0            |
|    6714 |            0 | 2334050437 |     3407268971 |        1603 |               80
|                    6 |                 10 |       1843 |        3030 |        4873 |
73900 |    3629130 |   3703030 |         0 |               3 | GET /mp3/rem |
|    6715 |            0 | 2334050437 |     3626103880 |        2007 |               80
|                    6 |                 19 |         16 |          13 |          29 |
1914 |      11402 |     13316 |         0 |               3 | GET /1748944 |
|    6716 |            0 | 2334050437 |     2334005284 |        4226 |             3389
|                    6 |                299 |       5069 |        5695 |       10764 |
419249 |     999231 |   1418480 |         0 |               3 | 0            |
|    6717 |            1 | 2334005284 |     2334050437 |        3389 |             1078
|                    6 |                281 |        276 |         280 |         556 |
64535 |      27484 |     92019 |         0 |               3 | 0            |
|    6718 |            0 | 2334050437 |     1158422324 |        1160 |             6667
|                    6 |                288 |         31 |          34 |          65 |
1509 |       3920 |      5429 |         0 |               3 | 0            |
|    6719 |            0 | 2334050403 |     2334052354 |       34528 |              139
|                    6 |                  9 |          3 |           0 |           3 |
180 |          0 |       180 |         0 |               3 | 0            |
|    6720 |            1 | 2334050421 |     2334050559 |         138 |              138
|                   17 |                146 |          5 |           0 |           5 |
1046 |          0 |      1046 |         0 |               3 | 0            |
|    6721 |            1 | 2334050509 |     2334050559 |         137 |              137
|                   17 |                 12 |          6 |           0 |           6 |
576 |          0 |       576 |         0 |               3 | 0            |
|    6722 |            1 | 2334050436 |     2334050403 |       61464 |               25
|                    6 |                  0 |         15 |          15 |          30 |
5360 |       1074 |      6434 |         0 |               3 |
         |
|    6723 |            1 | 3451652425 |     2334050437 |        5190 |             3394
|                    6 |                244 |          5 |           5 |          10 |
745 |        200 |       945 |         0 |               3 | 0            |
|    6724 |            1 | 2334050509 |     2334050559 |         138 |              138
|                   17 |                  0 |          1 |           0 |           1 |
```

Beispiel 1: TCP-Handshakingverfolgung

Fall 1: vollständiges Handshaking wurde erkannt

Etherreal-Screenshot:

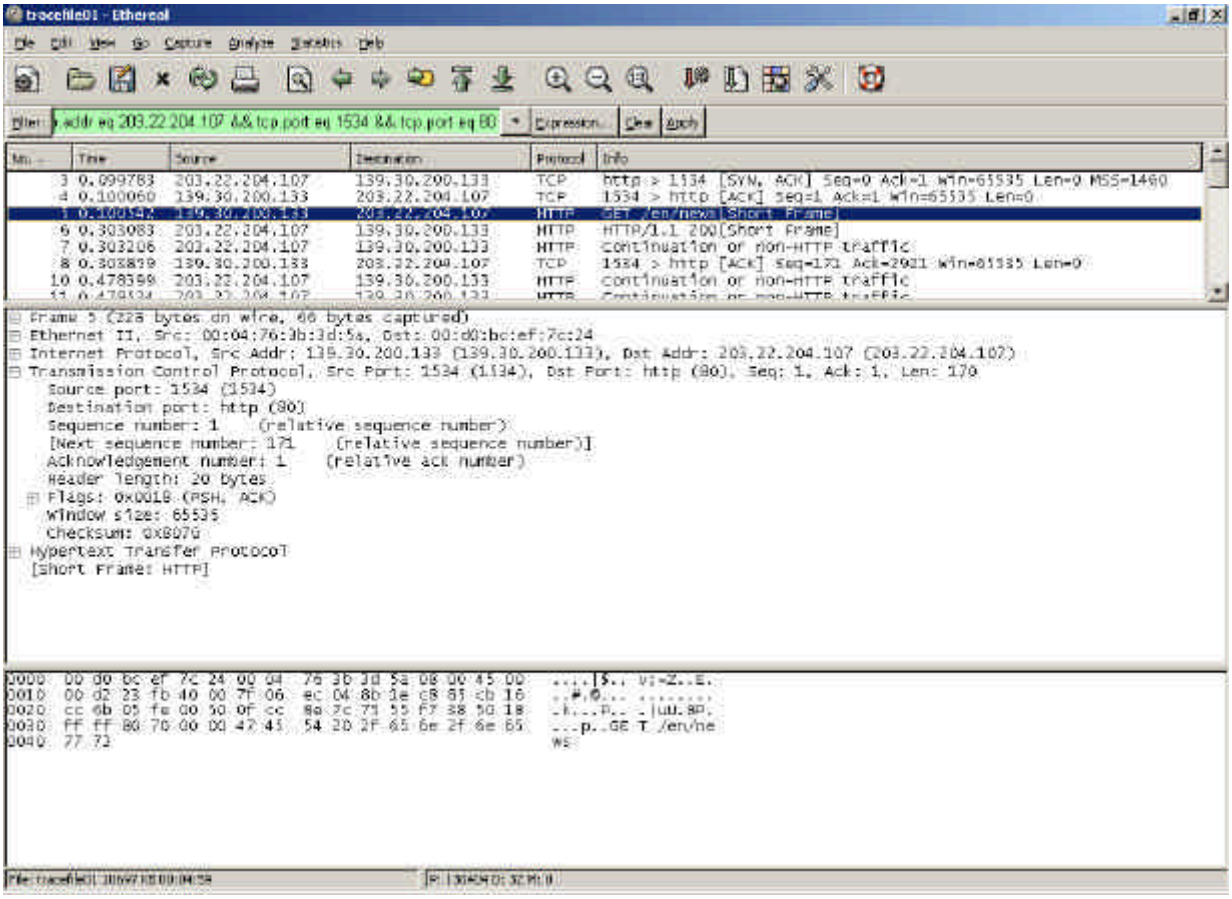


von *recordsneaker* ermittelte TCP-Payload: **GET /coremed**

Beispiel 2: TCP-Handshakingverfolgung

Fall 2: Handshaking ist erfolgt, jedoch erstes SYN-Paket wurde nicht erkannt

Etherreal-Screenshot:

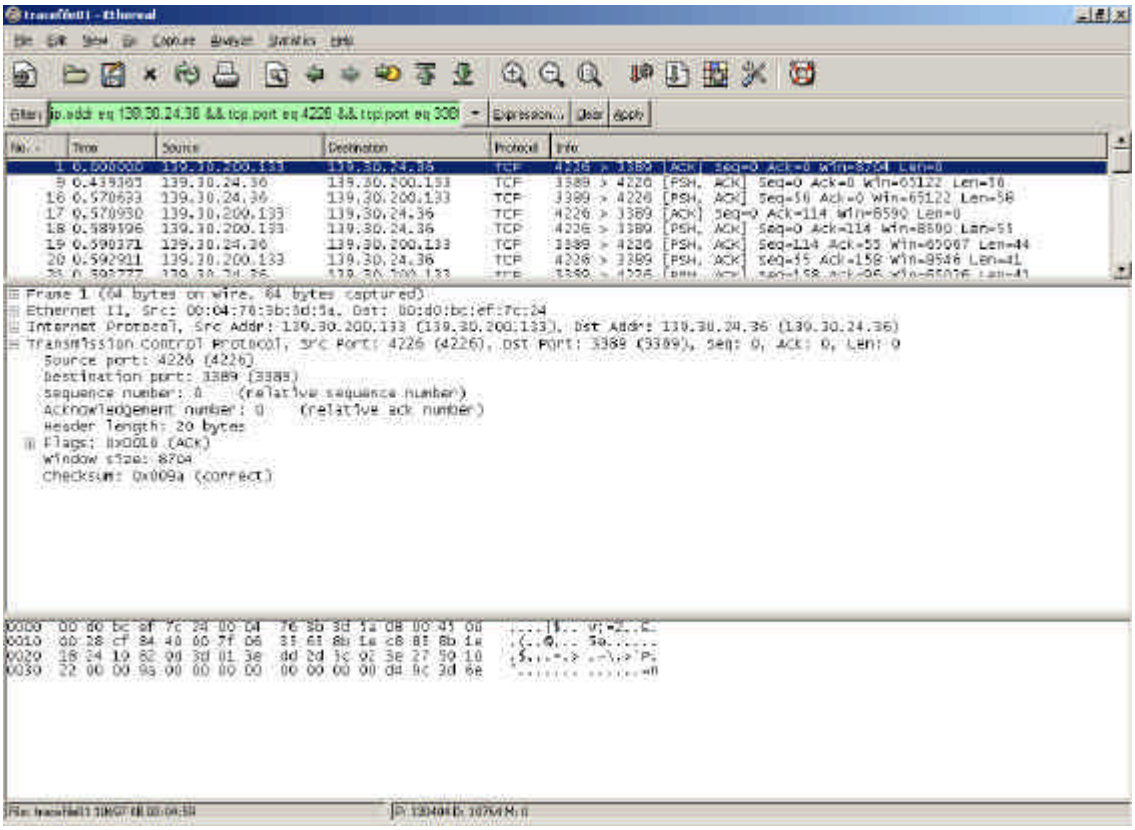


von *recordsneaker* ermittelte TCP-Payload: **GET /en/news**

Beispiel 3: TCP-Handshakingverfolgung

Fall 3: kein Handshaking ist erfolgt

Etherreal-Screenshot:



von *recordsneaker* ermittelte TCP-Payload: **keine**

Anhang D: recordsneaker README Datei

```
/*  ------------------------------------------------------------  */
/*   ####   ###   ###  #  # #   # # ##### #####                   */
/*   #   # #   # #   # # #  ##  # # #     #                       */
/*   #   # ##### #     ##   # # # # ####  ###                     */
/*   #   # #   # #  ## ##   #  ## # #     #                       */
/*   #   # #   # #   # # #  #   # # #     #                       */
/*   ####  #   #  ###  #  # #   # # #     #####                   */
/*  ------------------------------------------------------------  */
/*********************************************************************

 Project   : DAGKNIFE - univeral support tool for ENDACE.COM DAG boards
             Recordsneaker2 - flow based packet analysis tool for DAG streams
 Descript. : OSI Layer based analysis of DAG records
 Authors   : Thomas Kessler
             Christian Manthey
 Email     : thomas.kessler@gmx.net
             Christian.Manthey@gmx.de
 Created   : 15.11.2004
 Modified  : 22.02.2005
 TimeStamp : $Id: recordsneaker2.c,v 0.0.0.1 2004/22/02 23:18:02 skiros Exp $

*********************************************************************
 This program is free software; you can redistribute it and/or
 modify it under the terms of the GNU General Public License
 as published by the Free Software Foundation; either version
 2 of the License, or (at your option) any later version.
*********************************************************************/


Table of contents
-----------------

1. Program description
2. Source tree
3. Compiling the source
4. Program usage
5. Changes in version 2


1. Program description
----------------------

Recordsneaker2 is a flow based I packet analysis tool for input streams captured
by ENDACE DAG measurement boards written in C. It can process realtime or prestored
traffic.


2. Source tree
--------------
Recordsneaker2 source tree contains the following files:

/
              /include

                            buffer.h
                            config.h
                            dag_analysis.h
                            dag_frame.h
                            dag_rawio.h
                            dag_recordinfo.h
                            dag_types.h
                            flowfunc.h
                            icmp_header.h
                            ip_header.h
                            my_alloc.h
                            mysql_com.h
                            mysql_version.h
                            rec_io.h
                            tcp_header.h
                            udp_header.h

              /lib
```

```
                        libmysqlclient.a
                        librawio.a
                        librecio.a

                /progs

                        buffer.c
                        dag_analysis.c
                        flowfunc.c
                        readrecord.c
                        recordsneaker.c

                /raw_drivers

                        dag_drv.c
                        dag_drv.h
                        dag_rawio.c
                        file_drv.c
                        file_drv.h
                        skeleton_drv.c
                        skeleton_drv.h

                /rec_drivers

                        dag_recdrv.c
                        dag_recdrv.h
                        pcap_recdrv.c
                        pcap_recdrv.h
                        rec_io.c
                        skeleton_recdrv.c
                        skeleton_recdrv.h


3. Compiling the source
-----------------------

The used input libraries (rec_driver and raw_driver) are already compiled.

In recordsneaker2 source root directory:

Compiling:

        gcc -O2 -c progs/recordsneaker2.c progs/dag_analysis.c progs/buffer.c
           progs/readrecord.c progs/flowfunc.c -I include

Linking:

        gcc -o recordsneaker2 recordsneaker2.o dag_analysis.o readrecord.o buffer.o
           flowfunc.o -L lib -lrecio -lrawio -lmysqlclient -lz


The result is the file "recordsneaker2".


4. Program usage
----------------

Usage:

        recordsneaker2 [OPTION...] INPUTFILE

For program help:

        recordsneaker2 --help

For usage information:

        Usage: recordsneaker2 [-Dnstv?] [-A FLOWACT] [-b BUFSIZE] [-d DATABASENAME]
             [-h HOSTNAME] [-I FLOWINACT] [-l HASHSIZE] [-L LIMITTIME]
             [-o OUTPUTFILE] [-p PASSWORD] [-T TABLENAME] [-u USERNAME]
```

```
            [--flowact=FLOWACT] [--bufsize=BUFSIZE]
            [--databasename=DATABASENAME] [--db_mysql] [--hostname=HOSTNAME]
            [--flowinact=FLOWINACT] [--hashsize=HASHSIZE]
            [--limittime=LIMITTIME] [--nofile] [--output=OUTPUTFILE]
            [--password=PASSWORD] [--statistics] [--table]
            [--tablename=TABLENAME] [--username=USERNAME] [--verbose] [--help]
            [--usage] INPUTFILE

Options:

  -A, --flowact=FLOWACT      Set the flow activity time in seconds after a flow
                             is considered as finished, default is 600
  -b, --bufsize=BUFSIZE      Modify size of buffers used for storing flows,
                             default value is 400000
  -d, --databasename=DATABASENAME
                             MySQL database to connect to
  -D, --db_mysql             Produce output into MySQL database
  -h, --hostname=HOSTNAME    MySQL server to connect to, default is
                             "localhost"
  -I, --flowinact=FLOWINACT  Set the flow inactivity time in seconds after a
                             flow is considered as finished, default is 150
  -l, --hashsize=HASHSIZE    Modify size of hash table used for flow lists,
                             default value is 65535
  -L, --limittime=LIMITTIME  Read input data only for a certain time in
                             seconds, default is no time limit
  -n, --nofile               Produce no file output
  -o, --output=OUTPUTFILE    Output to OUTPUTFILE, default is ".\flowoutput"
  -p, --password=PASSWORD    Password corresponding to given user name for
                             MySQL server connection
  -s, --statistics           Produce program statistics after finishing flow
                             processing
  -t, --table                Produce table style flow output instead of comma
                             separated values
  -T, --tablename=TABLENAME  Table in given MySQL database to write data into
  -u, --username=USERNAME    User name for MySQL server connection, default is
                             "mysql"
  -v, --verbose              Produce verbose output
  -?, --help                 Give this help list
      --usage                Give a short usage message


Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.


5. Changes in version 2
-----------------------

- MySQL database output possible
- Output to file can be turned off
- Tracing of the payload of the first TCP packet after a sucessful TCP handshaking
       (output in MySQL database only)
- Output of source and destination IP in integer form (in CSV file format and MySQL
database)
- Tracing of the used DAG card interface number
```