

Bachelorthesis

Entwicklung einer Anwendung zur statistischen Analyse von TCP/IP-Netzwerken

Michael Fuckner
Matrikelnummer: 903722

26. Februar 2004

betreut durch Dr.-Ing. Helmut Dispert, FH Kiel,
Dipl.-Ing. (FH) Jan Kneschke, incremental GbR

Abstract

Das Thema dieser Bachelorarbeit ist die Analyse und Auswertung implementationsspezifischer Eigenarten bei der Kommunikation via TCP/IP. Bei Anfragen an einen Server sollen die Informationen über die Kunden protokolliert werden, um sie für Sicherheits- und Marketinganalysen aufzubereiten. Das Programm darf nicht in die Kommunikation zwischen Client und Server eingreifen und muss sich vollkommen passiv verhalten. Dies erspart dem Administrator umständliche Neukonfigurationen und verhindert, dass sich die Kunden abgeschreckt fühlen könnten. Das Programm muss modular aufgebaut sein, um es an zukünftige Anforderungen anzupassen. Die Resultate werden graphisch aufbereitet.

Danksagung

Eine Vielzahl von Leuten haben diese Bachelorthesis erst möglich gemacht. Ganz besonders bedanken möchte ich mich bei Herrn **Prof. Dr. Helmut Dispert** von der Fachhochschule Kiel für die hervorragende Betreuung meiner Arbeit, Herrn **Dipl.-Ing. (FH) Jan Kneschke** von incremental GbR für die vielen Ratschläge, **Jana Koch** (incremental GbR) für das Korrekturlesen, aufmunternde Worte und Kaffee und **Bsc Hendrik Scholz** für die Hilfe mit \LaTeX , die Bücher und die Beantwortung der damit verbundenen Fragen. Besonderer Dank gebührt **meinen Eltern**, die mir ein Studium erst ermöglichten.

Inhaltsverzeichnis

Einleitung	6
1. Problemdefinition	9
1.1. Ziele	9
1.2. Historische Entwicklung	12
1.3. Layer im TCP/IP-Modell	16
1.3.1. Host-to-network-Layer	16
1.3.2. Internet Layer	16
1.3.3. Transport Layer	19
1.3.4. Verbindungsaufbau	22
2. Problemanalyse	23
2.1. Aktives Fingerprinting	23
2.2. Passives Fingerprinting	24
2.2.1. Host-To-Network-Layer	24
2.2.2. Internet Layer (IP)	26
2.2.3. Transport Layer (TCP)	30
2.2.4. Application Layer	35
3. Implementation	37
3.1. Pcap	38
3.2. Parser - Monatliche Auswertung	42
3.3. Parser - Echtzeitauswertung	46
3.4. Ausgabe	48
3.5. Installation	51
3.6. Verwendete Software	52
Ausblick	55

Inhaltsverzeichnis

Literaturverzeichnis	56
A. Glossar	58
B. Tabellen- und Abbildungsverzeichnis	61

Einleitung

Der Grundstein für das Internet in seiner heutigen Form legte das amerikanische Militär in den 50er Jahren zur Zeit des kalten Krieges. Ziel war es, die Kommunikation zwischen den einzelnen militärischen Standpunkten zu optimieren und die Infrastruktur resistent gegen Störeinflüsse zu machen. Wichtig war, dass bei Ausfall einzelner Knoten die Kommunikation zwischen den verbleibenden Punkten nicht zusammenbricht. Die 1957 vom amerikanischen Verteidigungsministerium gegründete Forschungsabteilung *Advanced Research Projects Agency* (ARPA) beschäftigte sich mit der Suche nach Protokollen und Netztopologien, die diese Anforderungen erfüllten. Das Ergebnis der Forschung war ein Netz ohne zentrale Steuerinstitution, welches mit Datenpaketen konstanter Größe arbeitet. 1969 wurden die ersten Standpunkte, nämlich die Universitäten Los Angeles, Santa Barbara, Stanford und Utah miteinander vernetzt und bildeten so den Grundstock zum *ARPAnet*. Die ersten beiden Anwendungen, die für das *ARPAnet* zur Verfügung standen, waren *telnet* und *FTP*. *Telnet* steht für *telephone network* und ermöglicht die Nutzung entfernter Computer. *FTP* steht für *File Transfer Protocol* und dient dem Übertragen von Dateien auf entfernte Rechner. Diese wurden später in den so genannten Requests For Comment (RFC) mehrfach spezifiziert, aber erst in RFC854¹ (*telnet*, Mai 1983) beziehungsweise RFC959² (*FTP*, Oktober 1985) endgültig festgelegt.

Ray Tomlinson entwickelte 1971 *email* zur Übertragung von Textnachrichten über das Computernetz. 1973 kamen zu den 33 Standpunkten innerhalb der USA die ersten beiden Anbindungen aus Übersee hinzu (Großbritannien, Norwegen). Problematisch war, dass innerhalb des *ARPAnets* ein einheitlicher Kommunikationsstandard (*Network Control Protocol*, NCP) existierte, die neuen Rechner aber eigene Protokolle verwendeten. Im September 1973 wurde das *Transmission Control Protocol* (TCP) definiert, welches noch bis heute im Einsatz ist. Im Jahre 1973 betrug der Datenanteil an *email* im gesamten *ARPAnet* 75%.

Die Zahl der an das Internet angeschlossenen Rechner nahm im Laufe der Jahre rasant zu.

¹<http://www.faqs.org/rfcs/rfc854.html>

²<http://www.faqs.org/rfcs/rfc959.html>

Heutzutage sind nicht nur Universitäten vernetzt, sondern so gut wie jede mittelständische Firma ist an das globale Netzwerk angeschlossen. Firmen nutzen das Internet beispielsweise für die Organisation verteilter Prozessabläufe zwischen Filialen, Online-banking oder betreiben einen Onlineshop. Da die Preise, die für einen Internetzugang zu entrichten sind, stark zurückgegangen sind, besitzt heute fast die Hälfte³ der deutschen Privathaushalte einen Zugang zum Internet und 77% der 16-24-jährigen nutzen das Internet regelmässig. Für das Internet gibt es heute verschiedenste Anwendungen wie email, online-shopping, Internet-Banking, instant messaging oder Netzwerkspiele.

Als Folge dieser vielen Möglichkeiten steigt natürlich auch der Umfang der Anforderungen. Es ist für eine Firma unerlässlich, eine Internetpräsenz anzubieten, die stabil und jederzeit erreichbar ist. Jeder Ausfall schadet nicht nur dem Image, sondern verhindert auch die Annahme oder Bearbeitung von Aufträgen.

Um diese Sicherheit und Stabilität gewährleisten zu können, ist es wichtig, die Computersysteme zu analysieren, die diese Dienste anbieten und permanent mit dem Internet verbunden sind. Die sich ändernden Anforderungen müssen ausgewertet werden, damit die Dienste jederzeit mit der gleichen Geschwindigkeit angeboten werden können. Aber auch das Marketing hat Interesse an Informationen über die Kunden, die sich beispielsweise die Internetseite der Firma anschauen, um die Internetpräsenz an die verwendeten Browser anzupassen.

Die auf einem Server installierten Dienstprogramme protokollieren in sogenannten Logfiles, welche Verbindungen sie wann bearbeitet haben. Diese Logfiles enthalten für gewöhnlich den Zeitpunkt der Verbindung, die Adresse des anfragenden Rechners und welche Daten dabei ausgetauscht wurden. Die Intensität der protokollierten Ereignisse lässt sich meist konfigurieren. Dennoch ist die Menge der daraus gewonnenen Daten manchmal nicht ausreichend, weshalb andere Wege gefunden werden müssen, weitere Daten zu erhalten.

Es ist also wichtig, möglichst viele nutzerspezifische Daten zu sammeln und anonymisiert zu speichern. Heutzutage ist nahezu jeder Windows Rechner mit einer Firewall ausgerüstet, welche den Nutzer informiert, wenn man versucht, Informationen über den Wirtsrechner zu bekommen. Dies schreckt vermutlich viele Kunden ab und es ist daher ist es nicht zulässig, Anfragen an diesen zu verschicken. Man muß mit den Informationen auskommen, die über das Netzwerk übertragen werden und diese effektiver nutzen.

Bei der Überwachung von Computersystemen ist es wichtig, aus der Fülle der Informationen die relevanten herausgreifen. Hier wertet man primär die Informationen aus, die die Serverprogramme in die Logfiles geschrieben haben. Aber es gibt noch weitere Möglichkeiten. Ein

³<http://www.destatis.de/presse/deutsch/pm2003/p0511024.htm>

Ansatz ist das so genannte *Portscanning*. Dabei werden wahllos Anfragen an den zu prüfenden Rechner gestellt und geprüft, auf welche das Ziel antwortet. Nachteilig hierbei ist, dass die Kunden nicht gern mehr Daten als notwendig preisgeben.

Andererseits gibt es passives Fingerprinting. Hier werden keine Datenpakete an den Computer des Kunden geschickt. Durch die verschiedenen Implementationen des TCP/IP-Stacks sehen die verschickten Datenpakete je nach Betriebssystem anders aus. Wenn man nun die relevanten Informationen sammelt und mit einer Datenbank abgleicht, kann man mit hoher Wahrscheinlichkeit das verwendete Betriebssystem bestimmen.

Dort gibt es neben den offensichtlich anfallenden Informationen, welche von Serverprogrammen in Logfiles geschrieben werden auch zusätzliche Informationen, die vom Client ohne sein Wissen übertragen werden. Ziel dieser Arbeit ist es, diese zusätzlichen Informationen auszuwerten.

1. Problemdefinition

Zur Überwachung der Server, die ihre Dienste im Internet anbieten, gibt es viele Möglichkeiten. Die Aufgabe besteht darin, ein universelles Programm zu schaffen, welches die Performance des Servers nicht beeinträchtigt und universell für jede Art von Server einsetzbar ist. Es soll sich im Betrieb transparent für Administrator und Nutzer verhalten. Für die Firma incremental¹ ist es wichtig, dass das Programm sich problemlos in vorhandene Produkte wie den *localizer*² einbinden lässt und neben Linux auch auf andere UNIX-Derivate wie Solaris³ oder FreeBSD⁴ portierbar ist. Ein weiteres Kriterium für die Firma incremental ist die freie Verfügbarkeit der Software und die kommerzielle Nutzung aus freier Software geschaffener Produkte. Ferner müssen die gewonnenen Daten den Datenschutzrichtlinien entsprechen.

1.1. Ziele

Die erstellte Software muss folgende Anforderungen erfüllen:

Skalierbarkeit

Bei heute gebräuchlichen Servern ist es nicht unüblich, wenn 10000 Nutzer gleichzeitig einen Dienst nutzen. Hier ist es unerlässlich, dass die Software schnell arbeitet, damit der Server in seiner eigentlichen Tätigkeit nicht gebremst wird.

¹<http://www.incremental.de/>

²<http://www.incremental.de/products/localizer/>

³<http://www.sun.com/solaris/>

⁴<http://www.freebsd.org/>

Transparenz

Sowohl für den Nutzer, als auch den Administrator des Servers soll sich das Programm transparent verhalten. Dies spart umständliche Neukonfigurationen und Fehlersuche. Die ursprüngliche Art der Kommunikation zwischen Client und Server darf nicht verändert werden.

Mudularität

Es muss von Anfang an darauf geachtet werden, dass das Programm modular aufgebaut ist, so dass es problemlos um neue Analysemodule erweitert werden kann, wenn der Kunde diese verlangt.

Portabilität

Da die meisten Server Linux/ UNIX als Betriebssystem verwenden, sollte es auf mehreren Plattformen getestet sein und dort fehlerfrei funktionieren.

Freie Verfügbarkeit

Viele Softwareprodukte sind für die private Nutzung kostenlos. Gerade im Open-Source-Bereich ist es üblich, dass Software, die auf freien Programmen basiert, auch frei verfügbar sein muss und eine kommerzielle Nutzung ausgeschlossen ist. Solche Software darf nicht verwendet werden, da ein kommerzielles Produkt geschaffen werden soll. Eine Definition des Begriffes Open-Source und die als Open-Source anerkannten Software-Lizenzen findet man im Internet⁵.

Sicherheit

Das Programm soll auf Servern laufen, die direkt mit dem Internet verbunden sind. Da ein Ziel unter anderem die Verbesserung der Sicherheit dieser gefährdeten Systeme ist, darf an keiner anderen Stelle eine neue Sicherheitslücke durch gestartete Netzwerkdienste oder ähnliches geschaffen werden.

⁵<http://www.opensource.org/>

Verständliche Ergebnisse

Die gewonnenen Daten müssen als klare Ergebnisse dargestellt werden, die auch für Nicht-Techniker verständlich sind. Es hat sich gezeigt, dass durch farbige Visualisierung Daten schneller verinnerlicht werden können. Es sollen also als Resultat farbige Grafiken entstehen.

Datenschutzrichtlinien entsprechen

Es gibt zahlreiche Möglichkeiten, Kundendaten zu sammeln und miteinander zu verknüpfen. Wichtig ist, dass die Datenschutzrichtlinien eingehalten werden. In Deutschland gilt das *Telemediendatenschutzgesetz* (1997) ⁶ (TDDSG), welches ausdrücklich die Rechte des Anbieters einschränkt, sofern die gewonnenen Daten nicht der Abrechnung dienen. Das TDDSG sagt aus, dass über den Kunden erhobene Daten nur begrenzt gespeichert und verarbeitet werden dürfen. Zum Zwecke der Marktforschung oder Werbung dürfen Daten erhoben, aber aber nur anonymisiert verarbeitet nicht mit Nutzerdaten in Verbindung gebracht werden.

Auszug der relevanten Paragraphen aus dem TDDSG:

§ 6

[Nutzungsdaten]

(1) Der Diensteanbieter darf personenbezogene Daten eines Nutzers ohne dessen Einwilligung nur erheben, verarbeiten und nutzen, soweit dies erforderlich ist, um die Inanspruchnahme von Telediensten zu ermöglichen und abzurechnen (Nutzungsdaten).

Nutzungsdaten sind insbesondere

- a) Merkmale zur Identifikation des Nutzers,
- b) Angaben über Beginn und Ende sowie über den Umfang der jeweiligen Nutzung und
- c) Angaben über die vom Nutzer in Anspruch genommenen Teledienste.

(2) Der Diensteanbieter darf Nutzungsdaten eines Nutzers über die Inanspruchnahme verschiedener Teledienste zusammenführen, soweit dies für Abrechnungszwecke mit dem Nutzer erforderlich ist.

⁶http://www.datenschutz-berlin.de/recht/de/rv/tk_med/tddsg.htm

(3) Der Diensteanbieter darf für Zwecke der Werbung, der Marktforschung oder zur bedarfsgerechten Gestaltung der Teledienste Nutzungsprofile bei Verwendung von Pseudonymen erstellen, sofern der Nutzer dem nicht widerspricht. Der Diensteanbieter hat den Nutzer auf sein Widerspruchsrecht im Rahmen der Unterrichtung nach § 4 Abs. 1 hinzuweisen. Diese Nutzungsprofile dürfen nicht mit Daten über den Träger des Pseudonyms zusammengeführt werden.

(4) Der Diensteanbieter darf Nutzungsdaten über das Ende des Nutzungsvorgangs hinaus verarbeiten und nutzen, soweit sie für Zwecke der Abrechnung mit dem Nutzer erforderlich sind (Abrechnungsdaten). Zur Erfüllung bestehender gesetzlicher, satzungsmäßiger oder vertraglicher Aufbewahrungsfristen darf der Diensteanbieter die Daten sperren.

1.2. Historische Entwicklung

Heutzutage werden bei der Vernetzung von Computern meistens die Protokollfamilien TCP/IP über Ethernet eingesetzt. Alternative Protokolle wie *Decnet*, *Myrinet*, *SCI*, *Infiniband* oder *Token Ring* haben einen verschwindend geringen Marktanteil.

Netzwerkprotokolle sind in mehreren Schichten, so genannten *Layern* organisiert, bei der jede Schicht für eine bestimmte Aufgabe der Kommunikation zuständig ist. Es hat sich ein Modell durchgesetzt, welches die Theorie der Netzwerkkommunikation beschreibt- das *ISO/O-SI-Modell*. Dieses Modell verwendet sieben solcher Schichten (*physical, data link, network, transport, session, presentation, application*). Das in der Praxis gebräuchliche *TCP* hingegen arbeitet mit vier Schichten. In der Fachliteratur, wie dem *Computer Networks* [1] von Andrew S. Tanenbaum wird von fünf Schichten gesprochen. Die Übergänge sind demzufolge fließend.

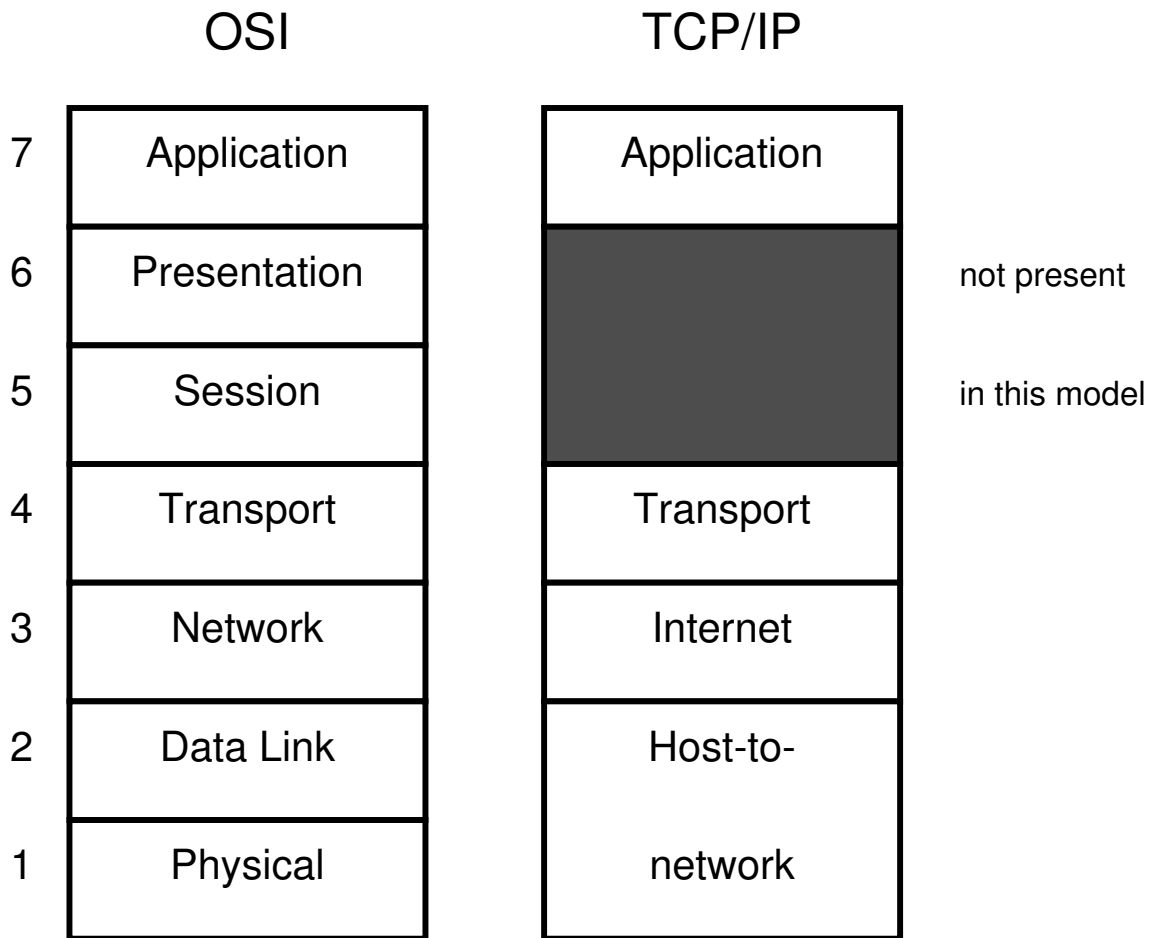


Abbildung 1.1.: Vergleich Layer ISO/OSI-TCP/IP

Bei der Kommunikation verwendet jede dieser Schichten ein eigenes Protokoll, welches die Daten mit einem Header versieht und diese dann zur Verarbeitung an die darunter liegende Schicht weitergibt. An der Empfängerseite verwendet man die gleichen Protokolle, um diese Header wieder zu entfernen. Dabei wird das Paket auf korrupte Daten geprüft, die eventuell durcheinander geratene Reihenfolge der Datenpakete wird wieder hergestellt oder doppelte Pakete werden aussortiert.

1. Problemdefinition

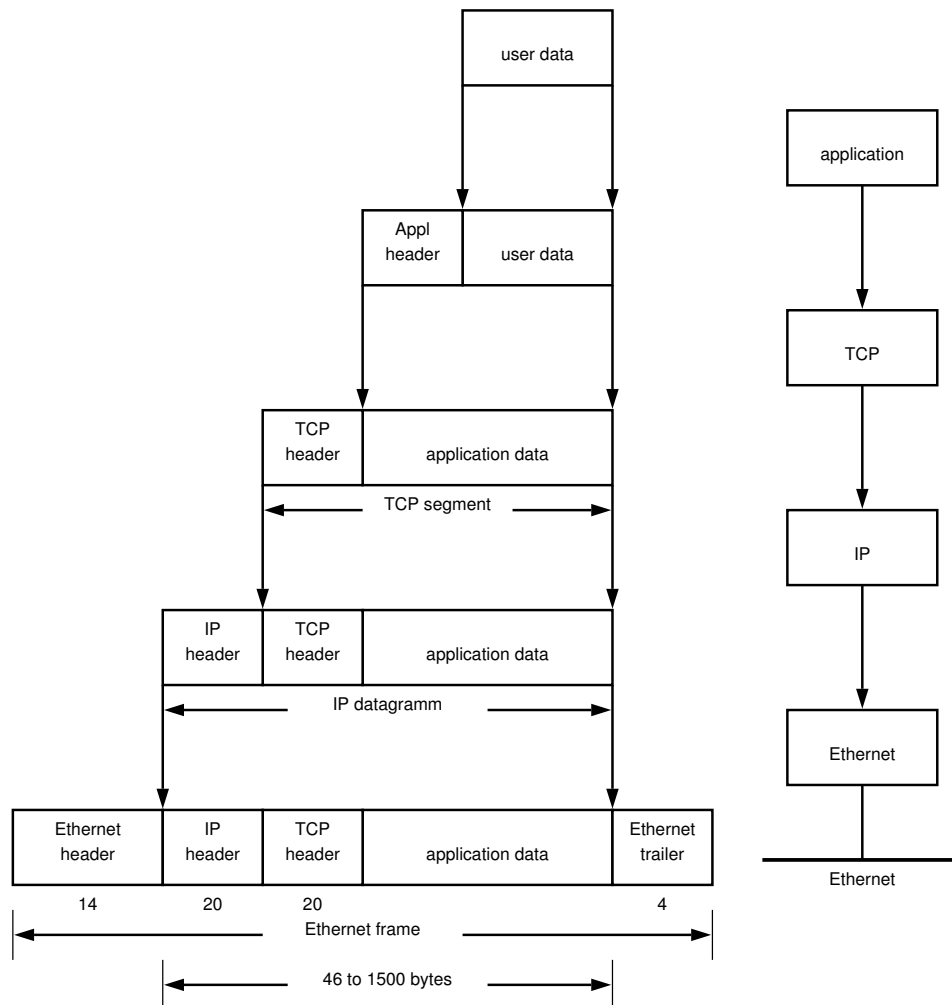


Abbildung 1.2.: Encapsulation

Die Protokolle wie TCP, IP oder ICMP sind in den so genannten RFC⁷s (*Request For Comment*) spezifiziert. Genauer gesagt beschreiben die Nummern 790-793 die Kommunikation per TCP/IP. Zusätzlich gibt es noch Ergänzungen, die später hinzugefügt wurden.

⁷<http://www.faqs.org/rfcs/>

Folgende RFCs waren oder sind weiterhin für die Kommunikation via TCP/IP relevant.

- 768: User Datagram Protocol (UDP)
- 790: Assigned Numbers
- 791: Internet Protocol (IP)
- 792: Internet Control Message Protocol (ICMP)
- 793: Transmission Control Protocol (TCP)
- 1042: Standard for the transmission of IP datagrams over IEEE 802 networks (beschreibt die Encapsulation)
- 1072: TCP extensions for long-delay paths (weitere TCP-Optionen)
- 1323: TCP Extensions for High Performance (ersetzt 1072, 1185)
- 1644: T/TCP – TCP Extensions for Transactions Functional Specification (T/TCP)
- 2700: Assigned Numbers (ersetzt 790)
- 2018: TCP Selective Acknowledgement Options (SACK)
- 3330: Special-Use IPv4 Addresses

Da die *Requests For Comment* nur Vorschläge und keine wirklichen Regeln sind, werden sie oft nicht strikt eingehalten und stellen eigentlich nur einen groben Rahmen für die Kommunikation dar. Damit erklären sich auch die vielen verschiedenen Implementationen und die daraus resultierenden Unterschiede im Aufbau der Pakete. Der Vorteil dieser RFCs ist, dass sie frei erhältlich sind und sich nur auf die Theorie beziehen. Das jeweils beschriebene Verfahren darf auf jeder Plattform und jedem Betriebssystem implementiert und eingesetzt werden.

1.3. Layer im TCP/IP-Modell

1.3.1. Host-to-network-Layer

Der Host-To-Network-Layer ist die unterste Schicht bei der Netzwerkkommunikation. Eine Verbindung auf dieser Ebene bedeutet, dass zwei Computer elektrisch miteinander verbunden sind und die Netzwerkkarten der beiden Maschinen sich gegenseitig sehen können. Auf dieser Ebene werden MAC-Adressen und der Ethernetheader übertragen. MAC steht für *Media Access Control* und beschreibt die Netzwerkkarte mit einer weltweit eindeutigen Nummer. Diese Nummer ist sechs Bytes lang und wird vom Hersteller der Netzwerkkarte vergeben. Die ersten drei Bytes beschreiben den Hersteller und die andere Hälfte ist eine vom Hersteller vergebene Seriennummer.

1.3.2. Internet Layer

Der Internet-Layer verwendet das *Internet Protokoll (IP)*, welches in RFC791 spezifiziert ist. Dieses Protokoll kümmert sich um die korrekte Zustellung der Datenpakete im TCP/IP-Netzwerk.

Ein IP-Header enthält folgende Informationen:

Version : Derzeit wird in Europa fast ausschliesslich IPv4 eingesetzt. IPv6 ist mittlerweile ausreichend erprobt, so dass man es einsetzen könnte. Doch solange ausreichend IP-Adressen vorhanden sind, ist der Einsatz noch nicht zwingend notwendig. Da Asien bei der Verteilung der IP-Adressen benachteiligt wurde, ist dort IPv6 oder NAT häufig im Einsatz, um diesen Mangel auszugleichen. IPv6 hat einen größeren Adressbereich, so dass mehr Rechner eine eindeutige Adresse bekommen können, bei NAT hingegen setzt ein Router die Adressen um. So kann ein Netzwerk an das Internet angeschlossen werden, das nur eine weltweit gültige IP-Adresse benutzt und intern Adressen verwendet, die nur in lokalen Netzen zugelassen sind. Eine Auflistung der verschiedenen IP-Bereiche findet sich in RFC3330⁸.

Header Length : Sie gibt an, wie groß der IP-Header ist und wo der Header des nächsten Protokolles beginnt.

⁸<http://www.faqs.org/rfcs/rfc3330.html>

1. Problemdefinition

4-bit version	4-bit header length	8-bit type of service (TOS)	16-bit total length (in bytes)		
16-bit identification			1-bit	1-bit	13-bits fragment offset
8-bit time to live (TTL)		8-bit protocol	16-bit header checksum		
32-bit source IP adress					
32-bit destination IP adress					

Abbildung 1.3.: IP-Header

- Type Of Service (TOS): Dieser Wert kann Optionen wie geringere Verzögerung, höheren Durchsatz, höhere Datensicherheit oder billigere Daten angeben. Dieses Feld wird nicht mehr verwendet.
- Total Length : Die Total Length gibt die Größe des gesamten Datenpaketes an.
- Identification (IPID) : Die IPID ist eine Seriennummer, die die IP-Pakete eindeutig identifiziert.
- Flags : Hier gibt es drei Bits, die für *Reserved Flag* (RF), *Don't Fragment* (DF) und *More Fragments* (MF) stehen.
- Fragmentation Offset : Dieser Wert wird im Falle von Fragmentierung genutzt und beschreibt die Verschiebung des Paketes in Bytes bezogen auf das Originalpaket. Wenn ein Paket fragmentiert wird, bezieht sich die *total length* nur auf das aktuelle Paket und nicht auf das Ursprungspaket.
- Time To Live (TTL) : Die *Time To Live* beschreibt die maximale Anzahl an Rechnern, die ein Paket durchlaufen darf, bevor es als unzustellbar verworfen wird. Jeder Rechner/ Router (genannt Hop) auf dem Weg verringert diesen Wert um einen Zähler. Ist der Wert Null erreicht, wird das Paket verworfen und eine ICMP-Fehlermeldung an den Absender geschickt.

1. Problemdefinition

- So wird der Absender informiert, dass der Zielrechner nicht erreichbar ist.
- Protocol : Dieser Wert bezieht sich auf den Inhalt des Paketes selbst. Gängige Werte sind hier eine (ICMP), zwei (IGMP) sechs (TCP) oder 17 (UDP).
- Header Checksum : Die Header Checksum ist eine Prüfsumme, die sich nur auf den Header bezieht.
- Source IP : Dieser Wert gibt die IP-Adresse des Rechners an, der das Paket verschickt hat. Er beschreibt den Rechner meist eindeutig.
- Destination IP : Die *Destination IP* steht für den Zielrechner, d.h. den Rechner auf dem das Programm laufen soll.
- Options : Optionen im IP-Header sind äußerst selten. Lediglich das *Internet Group Management Protocol* (IGMP), welches in RFC1112 spezifiziert ist, nutzt diese Optionen. Dieses Protokoll soll vorerst nicht weiter untersucht werden.

1.3.3. Transport Layer

Der Transport Layer nutzt in gebräuchlichen Internetanwendungen das in RFC793 spezifizierte *Transmission Control Protocol* (TCP). TCP stellt sicher, dass die übertragenen Datenpakete korrekt übertragen werden; gegebenenfalls ist dafür eine mehrfache Übertragung notwendig. Ein TCP-Header enthält folgende Informationen:

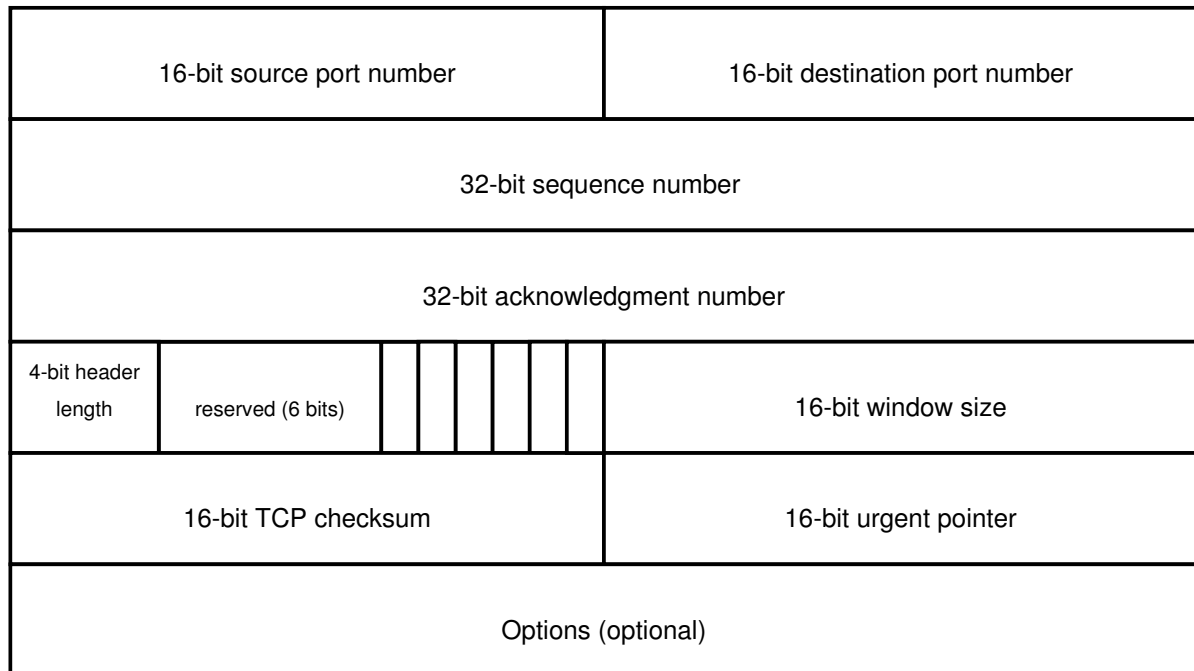


Abbildung 1.4.: TCP-Header

Source Port : Der *Source Port* beschreibt die Portnummer des sendenden Rechners.

Destination Port : Dieses Feld gibt die Portnummer an, auf der der sendende Rechner einen Dienst nutzen möchte.

1. Problemdefinition

- Sequence Number** : Diese Zahl beschreibt wie viele Bytes schon bei der aktuellen Verbindung übertragen wurden. Die *Sequence Number* ist ein 32bit unsigend Wert, der bei $2^{32} - 1$ überläuft.
- Acknowledgment Number**: Dieser Wert ist gültig, wenn das ACK-Flag gesetzt ist. Da die übertragenen Bytes gezählt werden und dieser Wert immer mit übertragen wird, enthält dieses Feld die Sequence Number des letzten empfangenen Paketes +1, um den Sender die fehlerfreie Übermittlung mitzuteilen.
- Header Length** : Beschreibt die Länge des TCP-Headers, der Wert ist primär von der Anzahl der Optionen abhängig. Die maximale Länge bei diesem 4bit Zähler beträgt 15 Doppelworte (60 Byte).
- Flags** :
- URG (*urgent*)- Die enthaltenen Daten sind dringend, der Urgent-Pointer ist gültig
 - ACK (*acknowledge*) - Das ACK-Flag ist eine positive Bestätigung der vorangegangenen Anfrage. Dies ist entweder ein Paket mit gesetztem SYN. In diesem Fall ist im Antwortpaket SYN + ACK gesetzt. Anderenfalls wird damit der Empfang eines Datenpaketes bestätigt.
 - PSH (*push*)- Das Push-Flag veranlasst die Gegenstelle dazu, Daten so schnell wie möglich zu übertragen. Es ist dann kein anderes Flag gesetzt.
 - RST (*reset*)- Dieses Flag ist gesetzt, wenn beispielsweise auf dem angefragten Port kein Dienst lauscht oder eine Firewall die Verbindung ablehnt.
 - SYN (*synchronize*)- Dieses Flag ist gesetzt, wenn der Client das erste Paket an den Server schickt, um eine Verbindung aufzubauen.
 - FIN (*finish*)- Der Clientrechner will die Kommunikation abbrechen.
- WindowSize** : Die *WindowSize* sagt aus, wie viele Daten maximal ohne Zustimmung des Empfängers gesendet werden dürfen. Dieser Wert kann durch einen Faktor, der durch die Option *WindowScale* (Typ3) beschrieben ist, vergrößert werden. Die *WindowSize* ändert sich mit

1. Problemdefinition

jedem Paket. Der Faktor ist fest und wird beim Handshake bestimmt. Es kann bei einer Verbindung für Sender und Empfänger verschiedene Wscale-Werte geben.

- TCP Checksum : Die *TCP Checksum* ist eine Prüfsumme, die aus dem TCP Header und den TCP-Daten berechnet wird.
- Urgent Pointer : Dieser Zeiger ist nur gültig, wenn das URG-Flag gesetzt wurde. Er gibt an, wie viele Bytes/ Daten/ Pakete mit dringenden Daten noch folgen werden.
- Options : Hier gibt es verschiedene Optionen, keine von ihnen ist Pflicht. Heutzutage verwenden jedoch alle Betriebssysteme wenigstens die *Maximum Segment Size* (MSS), welche die maximal in einem Zug zu übertragende Paketgröße angibt.

1.3.4. Verbindungsaufbau

Der Verbindungsaufbau bei TCP/IP läuft in drei Schritten ab.

1. Der Client-Rechner schickt an den Server-Rechner ein Datenpaket, in dem das SYN-Flag gesetzt ist.
2. Der Server bestätigt, dass er das Paket mit der Sequenznummer empfangen hat und sendet seine eigenen unterstützten Optionen, wobei SYN und ACK-Flags sind gesetzt. Sollte der Server nicht mit dem Client kommunizieren wollen, weil auf diesem Port kein Dienst lauscht oder eine Firewall dies nicht zulässt, schickt dieser ein Paket mit gesetztem RST-Flag. Damit informiert er den Client, dass seine Verbindung nicht erwünscht ist.
3. Der Client schickt als Bestätigung ein Paket mit ACK, dass er das Paket fehlerfrei empfangen hat. Die Verbindung ist nun aufgebaut und der reguläre Datenaustausch beginnt.

Dieser, bei TCP/IP übliche, dreistufige Verbindungsaufbau (genannt *three-way handshake*) ist in der folgenden Skizze noch einmal graphisch dargestellt:

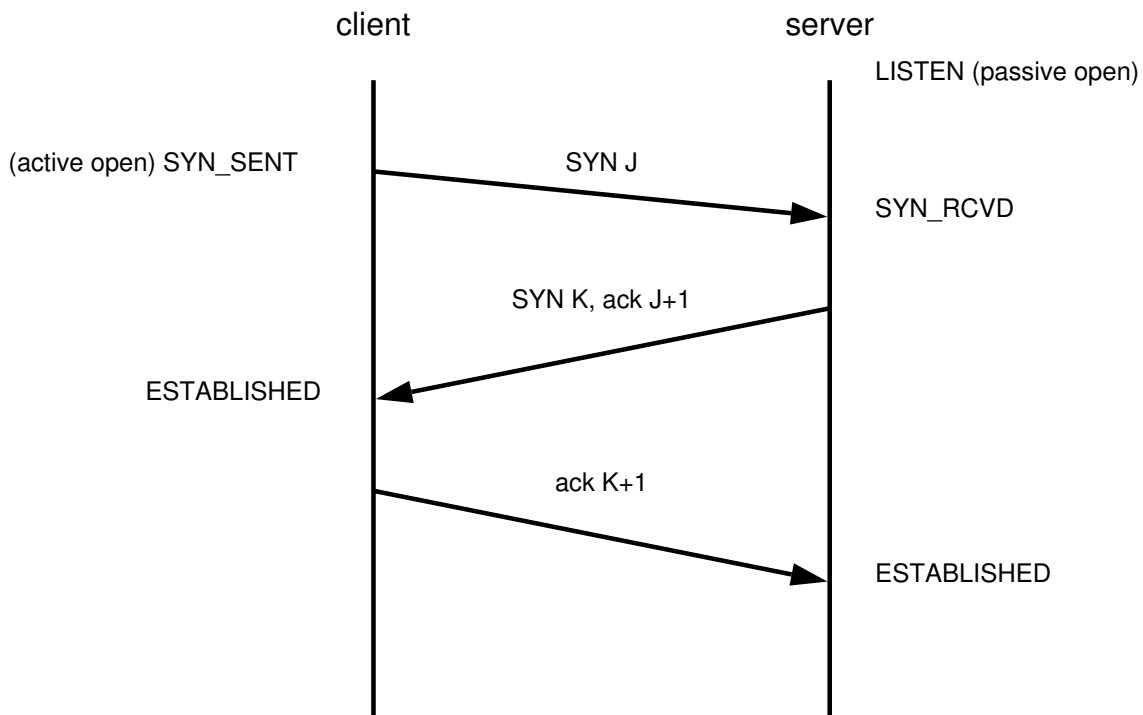


Abbildung 1.5.: TCP/IP Handshake

2. Problemanalyse

Es gibt mehrere Möglichkeiten, sich über die Kommunikation im Netzwerk zu informieren. Klassisch ist die Auswertung von Logfiles. Jede Serverapplikation schreibt im Normalfall den Status des Dienstes in eine Textdatei. Meist sind dies Informationen über den Zeitpunkt der Datenübertragung durch die jeweilige IP. Zudem werden erfolgreiche beziehungsweise fehlgeschlagene Authentisierungen -eventuell auch noch die von den Clientprogrammen übertragenen Versionsbezeichnungen- protokolliert. Im Normalfall ist einstellbar, wie intensiv diese Daten aufgezeichnet werden.

Oft sind die Daten nur sehr vage, die Meldungen der Clients sind nicht eindeutig und können durch Proxies manipuliert werden. Was tun, wenn man nun mehr Daten benötigt, als der Client freiwillig bereitstellt? Dazu gibt zwei primäre Ansätze:

2.1. Aktives Fingerprinting

Bei diesem Verfahren werden aktiv Datenpakete an den zu untersuchenden Rechner geschickt und seine Reaktionen ausgewertet. Die Bekannteste von ihnen ist sicherlich das sogenannte *portscanning*, bei dem bei vielen Ports auf dem Zielrechner geprüft wird, ob dort ein Dienst antwortet. Es gibt mehrere Programme, die Portscans durchführen können, sehr weit verbreitet ist das Programm *nmap*¹. Dieses Programm kann neben Portscans auch speziell präparierte Pakete an den Zielrechner schicken und anhand dessen Reaktion diesen klassifizieren. Mithilfe dieser Daten versucht *nmap*, Betriebssysteme zu bestimmen, wie in dem Artikel auf [insecure.org](http://www.insecure.org)² beschrieben. Dies versucht *nmap*, indem es zum Beispiel ein Paket mit gesetztem FIN-Flag schickt. Es versucht also eine Verbindung abzubauen, die gar nicht da ist (FIN-Probe). Zwar sollte so ein Paket laut RFC793 verworfen werden, viele Betriebssysteme antworten dennoch auf diese Anfrage. Andere Tests beobachten die Entwicklung der IPID, der Sequenznummer oder der Timestamps.

¹<http://www.insecure.org/nmap/>

²<http://www.insecure.org/nmap/nmap-fingerprinting-article.html>

Portscanning lässt sich mit dem Schleichen um ein Haus vergleichen, bei dem man an allen Türen rüttelt, um sich Zutritt zu verschaffen. So ein Verhalten schreckt natürlich potenzielle Kunden ab und kann in diesem Fall nicht verwendet werden.

2.2. Passives Fingerprinting

Ein anderer Ansatz ist das passive Fingerprinting. Dort werden keine Daten verschickt, es wird einfach auf dem Netzwerk gelauscht, welche Daten übertragen werden. Der Vorteil an passivem Fingerprinting ist, dass es absolut unsichtbar im Netzwerk ist, da es keine eigenen Pakete verschickt. Es hat aber den Nachteil, dass man keine Daten abfragen kann, sondern nur zur Verfügung hat, was im Netzwerk vorbeischwimmt. Da die Kommunikation in den RFCs spezifiziert ist, welche nur grobe Vorgaben machen, fällt die Implementation überall anders aus. Genau an dieser Stelle kann man ansetzen: die Eigenarten der übertragenen Pakete werden ausgewertet. Da die Spezifikationen der Protokolle TCP und IP nicht wirklich bindend sind, gibt es viele Möglichkeiten, die Kommunikation zu implementieren.

Wichtig ist es, das erste Datenpaket, das SYN-Paket zu analysieren, da nur dort die TCP-Option MSS, die *Maximum Segment Size* und die Angabe, dass der Computer das *Sequential Acknowledge*-Verfahren (SACK) unterstützt, übertragen wird. Es wäre wünschenswert, mehrere Pakete zu analysieren, aber dafür benötigt man eine echte *TCP-State Machine*, man muss also alle Zustände, die bei der TCP-IP-Kommunikation auftreten können, nachbilden. Diese Aufgabe sprengt den Rahmen dieser Arbeit bei weitem.

2.2.1. Host-To-Network-Layer

Im Host-to-Network-Layer existieren folgende Informationen: Mac-Adressen und Informationen, dass es sich um Ethernet handelt:

```
e0 18 dc 42 70 00 0a e6 2f e9 77 08 00
```

Hier erkennt man die MAC-Adressen 00:e0:18:cd:42:70 (Ziel) und 00:0a:e6:2f:e9:77 (Quelle) der beiden beteiligten Computer. Dahinter folgt die 08:00. Dieser Wert kann für die Länge stehen, wenn das in RFC1042 spezifizierte IEEE802.3/802.3- Verfahren zum Einsatz kommt. Wird hingegen Ethernet verwendet (RFC894), so steht dieses Datum für den Typ der Daten.

Jeder Hersteller von Netzwerkkarten hat einen Wertebereich, den er für die weltweit eindeutige Adresse seiner Netzwerkkarten verwenden kann. Bei normalen Computern sagt diese Adresse natürlich nicht viel aus, abgesehen davon, dass der Rechner eine Netzwerkkarte der Firma 3COM, Realtek, Intel oder eine andere verwendet. Bei seltenen Computern jedoch, ist diese Adresse um so aussagekräftiger. Diese Adressen werden von der IEEE³ vergeben. Dieses *Institute of Electrical and Electronics Engineers* beschäftigt sich unter anderem mit der Normung elektrotechnischer Standards. Die von der IEEE vergebenen Nummern sind teilweise in RFC1700 aufgelistet. Aber auch die *Internet Assigned Numbers Authority*⁴ (IANA) vergibt Ethernetadressen. Es gibt im Internet sowohl bei der IEEE⁵, als auch bei der IANA⁶ eine Liste, wie diese Adressen vergeben sein sollten. In der Praxis wird davon jedoch häufig abgewichen. Eine brauchbare Suchmaschine für MAC-Adressen findet im Internet⁷. Dort erfährt man, dass Mac-Adressen, die mit 00:03:ba beginnen, von der Firma Sun vergeben werden. Dieses Verfahren eignet sich hervorragend, seltene Hardware sicher zu erkennen, da die Werte eindeutig sind. Allerdings lassen sich diese Werte sehr einfach fälschen. So setzt beispielsweise unter Linux/ UNIX der Befehl `ifconfig fxp0 ether 01:02:03:04:05:06` die Mac-adresse des Interfaces fxp0 auf den angegebenen Wert. Auch wird dieser Wert nur im lokalen Netz übertragen und eignet sich daher nicht zur Identifikation fremder Rechner im Internet. Die MAC-Adresse wird im Ethernet-Header übertragen, lässt sich aber auch in der Konsole folgendermaßen anzeigen:

Wenn nun als Netzwerktyp nicht reines Ethernet zum Einsatz kommt, sondern eine DSL Verbindung, sehen die auf unterster Ebene übertragenen Daten folgendermaßen aus:

```
00 00 02 00 00 00 00 00 00 00 00 00 00 00 08 00
```

Diese Daten kann man in folgende Einzelkomponenten zerlegen:

- 00 00 == sll_pkttype
- 02 00 == sll_hatype
- 00 00 == sll_halen
- 00 00 00 00 00 00 00 00 == sll_addr

³<http://www.ieee.org/>

⁴<http://www.iana.org/>

⁵<http://standards.ieee.org/regauth/oui/oui.txt>

⁶<http://www.iana.org/assignments/ethernet-numbers>

⁷http://coffer.com/mac_find/

2. Problemanalyse

- 08 00 == sll_protocol

```
v440 # ifconfig ce0 | grep ether
      ether 0:3:ba:52:d6:dd
v440 # uname -snrvmi
SunOS v440 5.9 Generic_112233-10 sun4u SUNW,Sun-Fire-V440
```

Ausgabe der Netzwerkadresse auf einer Sun Fire V400 unter SunOS5.9 (Solaris9)

Ergebnis der Suchabfrage bei http://coffer.com/mac_find/

Search results for "0003ba"

```
MAC Address
Prefix          Vendor
0003BA          Sun Microsystems
```

2.2.2. Internet Layer (IP)

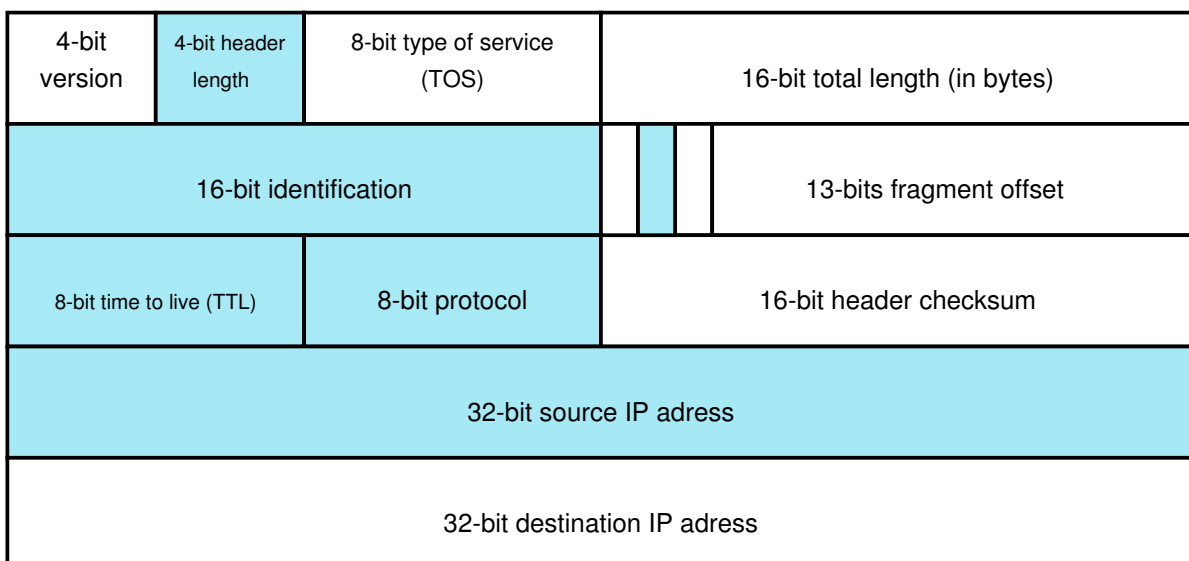


Abbildung 2.1.: IP-Header, relevante Partien markiert

2. Problemanalyse

Auf Ebene der Internet-Schicht arbeitet das sogenannte *Internet Protocol* und überträgt im Header folgende Informationen:

- Version : Da heutzutage in Europa fast nur IPv4 im Einsatz ist, sollte auch nur dieses Protokoll analysiert werden. Eine Erweiterung dieses Programmes auf IPv6 ist aber relativ einfach möglich.
- Header Length : Sie gibt an, wie groß der IP-Header ist und wo der Header des nächsten Protokolles beginnt. Aus diesem Wert lassen sich keine Schlüsse ziehen.
- Type Of Service (TOS): Da dieses Feld in der Regel nicht genutzt wird, macht es keinen Sinn, seinen Wert zu analysieren.
- Total Length : Die Total Length gibt die Größe des Gesamtpaketes an. Da nur SYN-Pakete betrachtet werden, ist dieser Wert irrelevant, da im Datenfeld kaum Informationen stehen.
- Identification (IPID) : Die IPID ist eine Seriennummer, die die IP-Pakete identifiziert. Bei manchen Betriebssystemen hat die IPID beim Verbindungsaufbau (im SYN-Paket) den Wert Null. Daran erkennt man z.B. Linux 2.2/2.4 und FreeBSD 5.2-CURRENT. Beobachtet man mehrere Pakete, kann man anhand der Änderung des Wertes Rückschlüsse auf das Betriebssystem ziehen. Hier existieren auch wieder verschiedene Möglichkeiten der Implementation. Bei einigen Betriebssystemen wird der Wert von Paket zu Paket um eins erhöht (Windows), bei anderen um den Wert 255 (htons(1), Maschinen deren Prozessor im Big Endian Format rechnen) oder man verwendet echte Zufallszahlen, wie es auf Sicherheit optimierte Betriebssysteme wie OpenBSD tun. Wenn man die IPID über einen längeren Zeitraum beobachtet, kann man nicht nur NATs und Proxies erkennen, sondern auch dahinter schauen.
- Manche Betriebssysteme verwenden einen eigenen Zähler für Pakete des Typs TCP, UDP und ICMP.
- Flags : Hier gibt es drei Bits, die für *Reserved Flag* (RF), *Don't Fragment* (DF) und *More Fragments* (MF) stehen. Das RF-Flag ist reserved, darf also nicht gesetzt werden. Wenn das DF-Flag gesetzt ist, darf das Paket unter keinen Umständen fragmentiert werden. Wenn also ein Paket mehr Daten enthält,

als die MSS groß ist und dieses Flag gesetzt ist, wird das Paket verworfen. Der Rechner, der das Paket verwirft, schickt eine ICMP-Fehlermeldung an den Sender, um ihm mitzuteilen, dass er das Paket noch einmal übertragen muss.

Das MF-Flag ist gesetzt, wenn noch weitere Fragmente folgen. Bei dem letzten Paket der Serie ist dieses Flag nicht gesetzt. Für die Analyse ist nur das DF-Flag relevant, da SYN-Pakete nicht groß genug sind, um fragmentiert zu werden. Lediglich das DF ist bei manchen Implementationen gesetzt und bei manchen nicht. Dieses ist ein zu protokollierendes Charakteristikum.

Fragmentation Offset : Da die SYN-Pakete zu klein sind, um fragmentiert zu werden, ist dieser Wert für die Analyse irrelevant.

Time To Live (TTL) : Man kann anhand dieses Wertes einfache Aussagen darüber treffen, wie weit der Rechner entfernt ist, da jeder Hop den Zähler dekrementiert. Ein Rechner, der zwanzig Hops entfernt ist, ist meist geographisch weiter entfernt, als einer, der drei Hops entfernt ist.

Man kann versuchen, den Weg rückwärts zu dem entsprechenden Rechner zu gehen, wenn man das Programm *tracert* (unter Windows *tracert*) nutzt. Dieses Programm versendet der Reihe nach Pakete an den Zielrechner mit einer TTL von eins, zwei usw., bis die TTL ausreichend groß ist und das Datenpaket bei dem Zielrechner ankommt. Es ist dabei allerdings nicht garantiert, dass jedes Paket denselben Weg nimmt und dass der Weg, den die Pakete gekommen sind, auch der Weg ist, den die mit *tracert* verschickten Pakete nehmen. Das Problem an der TTL ist, dass man diesen Wert nicht direkt auslesen kann, sondern schätzen muss, mit welcher TTL das Paket verschickt wurde. Für eine genaue geographische Auswertung existiert im Sortiment der Firma incremental bereits das Produkt *localizer*⁸, so dass hier eine genauere Auswertung unnötig ist.

Die TTL ist sehr aussagekräftig, da es verschiedene Klassen von Betriebssystemen gibt:

- 32: Windows 95
- 64: Free-, Net- oder OpenBSD

⁸<http://www.incremental.de/products/localizer/>

2. Problemanalyse

- 128: Windows 2000/ XP
- 255: Solaris

Protocol	: Da hier nur TCP und eventuell später ICMP analysiert werden soll, ist es zwar wichtig, diesen Wert zu protokollieren, Ergebnisse kann man daraus allerdings nicht gewinnen.
Header Checksum	: Die Header Checksum ist eine Prüfsumme, die sich nur auf den Header bezieht. Sie ist beim Fingerprinting nicht von Bedeutung, da sie nur aussagt, ob das Paket heil empfangen wurde, oder, ob eine Störung vorlag.
Source IP	: Dieser Wert beschreibt die IP-Adresse des Rechners, der das Paket verschickt hat und kennzeichnet den Rechner meist eindeutig. Es kann allerdings sein, dass mehrere Rechner durch <i>Network Address Translation</i> auf eine externe IP-Adresse abgebildet werden.
Destination IP	: Die <i>Destination IP</i> beschreibt den Zielrechner, d.h. den Rechner auf dem das Programm laufen soll. Da das Betriebssystem bekannt ist und die meisten Rechner nur eine IP-Adresse besitzen, wird dieses Datum als unwichtig verworfen.

2.2.3. Transport Layer (TCP)

Auch der TCP-Header enthält implementationsspezifische Eigenarten, die in diesem Kapitel erläutert werden.

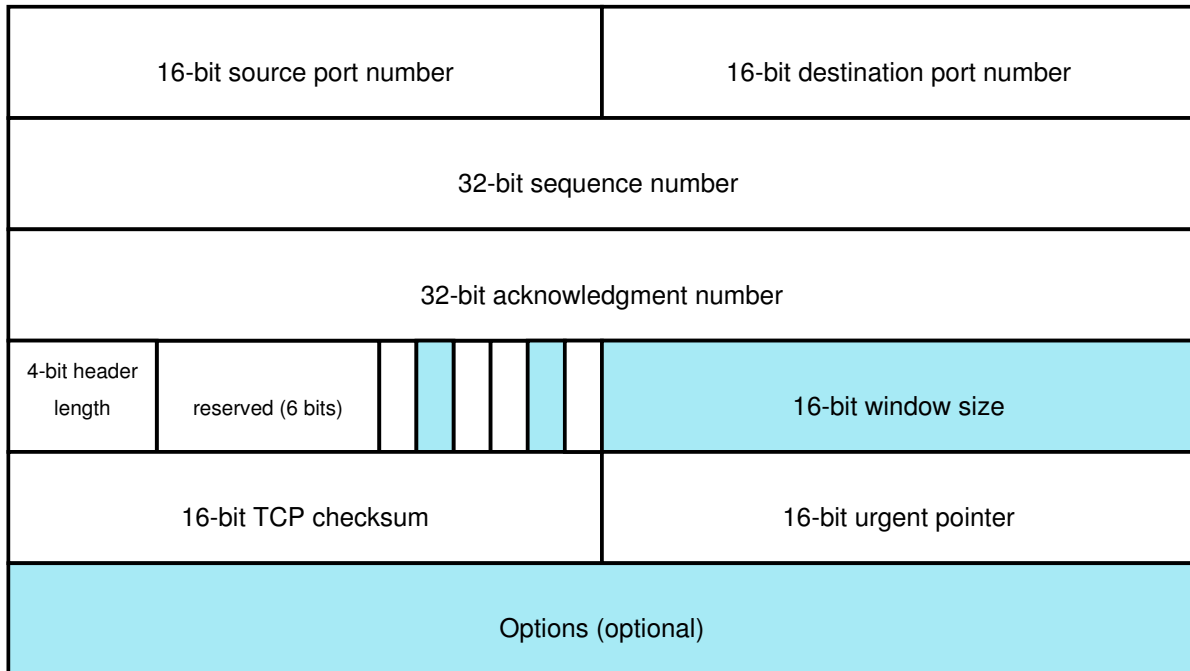


Abbildung 2.2.: TCP-Header, relevante Partien markiert

- Source Port : beschreibt die Portnummer des sendenden Rechners
- Destination Port : beschreibt die Portnummer, auf der der sendende Rechner einen Dienst nutzen möchte.
- Sequence Number : bezieht sich auf die aktuelle Verbindung und dient der fehlerfreien Kommunikation, für diese Aufgabe nicht von Bedeutung
- Acknowledgment Number: wie *Sequence Number*

2. Problemanalyse

- Header Length : ist durch die Anzahl der Optionen festgelegt; da diese speziell analysiert werden, ist dieses Datum redundant.
- Flags : URG: diese Daten sind dringend, vereinzelt setzen Windowsrechner dieses Flag im SYN-Paket, es ist nicht reproduzierbar.
SYN: Dieses Flag sagt aus, dass der Rechner eine Verbindung aufbauen will. Es ist im ersten und zweiten Paket der TCP/IP-Kommunikation gesetzt. Hier werden nur Pakete analysiert, die das SYN und nicht das ACK-Flag gesetzt haben, es soll nur das erste Paket der Kommunikation verwendet werden.
- WindowSize : Dieser Wert ist charakteristisch dafür, wie viele Daten das Betriebssystem annehmen kann. Er kann um die Option Wscale (Typ 3) erweitert werden, welche am Anfang der Verbindung ausgehandelt wird. Je kleiner der Rechner, desto kleiner ist auch seine gesendete Windowsize.
- Tcp Checksum : Dieses Datum ist für diese Betrachtung uninteressant, da sie aus dem Header und den Daten errechnet wird und nur angibt, ob das Paket heil übertragen wurde.
- Urgent Pointer : Da einige Windowsversionen unter nicht reproduzierbaren Umständen das URG-Flag setzen, kann man auch aus diesem Wert keine sinnvollen Rückschlüsse ziehen.
- Options : Von diesen Optionen ist keine Pflicht. Heutzutage verwenden alle Betriebssysteme wenigstens die *Maximum Segment Size* (MSS), welche angibt, wie große Pakete ohne Fragmentierung übertragen werden können. Es ist empfehlenswert, diesen Wert anzugeben, da sonst die Paketgröße den Standardwert von 576 Bytes annimmt. Mehr Daten pro Paket bedeutet weniger Arbeit für die Netzwerkkarte bei der Übertragung der gleichen Menge an Daten. Typische Werte liegen bei 1260-1460, was bedeutet, dass in einem Paket zwei- bis dreimal so viele Daten übertragen werden können. Deshalb ist diese Option sehr zu empfehlen.
Der genaue Wert ist primär von der Art der Anbindung abhängig. Über DSL angeschlossene Rechner verwenden hier beispielsweise 1452, damit die verschiedenen Header⁹ (PPP (8), IP (20), TCP

⁹Es werden nur die Header fester Größe gezählt, nicht die Optionen

(20)) und die Daten in einen Ethernet-Frame mit 1500 Bytes passen. Man kann aber auch über die Reihenfolge der Flags Rückschlüsse ziehen. Die meisten Windowssysteme verwenden die Optionen MNNO. Hierbei ist anzumerken, dasss Windows 98 beispielsweise die Optionen MNNO sendet, als Wert für die MSS jedoch 536. Man könnte diese Option also weglassen.

Wenn auf dem System ein VPN-Client installiert ist, kann dieser Wert auch heruntergesetzt werden. Besonders bei Windows 98, da dieses Betriebssystem bei verschiedenen Netzwerkverbindungen nur eine Einstellung kennt und der geringere Wert eingetragen wird.

Anhand der Timestamps und deren Änderung kann man auch etwas über die Betriebssysteme aussagen. Wenn man die Zeit misst (und annimmt, dass die Laufzeit der Pakete in etwa konstant ist), kann man die Änderung der Timestamps für Analysen verwenden. Ein Linux 2.4 beispielsweise inkrementiert den Zähler alle 10ms und ein Linux 2.6 jede ms. Dadurch könnte man bei gleicher TCP-Ausgabe diese beiden Betriebssysteme unterscheiden.

Aber mit dem Timestamp kann man, genau wie mit der IPID, auch durch *Network Address Translations* (NAT) hindurchschauen. Dies erfolgt bei Zugriffen verschiedener Rechner auf unseren Server mit komplett verschiedenen Timestamps. In diesem Fall kann man versuchen, diese in Relation zu setzen. Man wird bei mehreren Zugriffen mehrerer Rechner mit einer IP die einzelnen Zeitbasen der Quellrechner ermitteln können. Wenn jedoch ein HTTP-Proxy wie zum Beispiel squid¹⁰ den Zugriff auf das Internet regelt, bekommt man nur die timestamps dieses Rechners übermittelt.

Für die gesamte Analyse ist es allerdings notwendig, mehrere Pakete zu beobachten; dies ist eine Analyseoption für die Zukunft.

Gebräuchliche Optionen sind:

End: Typ: 0, keine Längenangabe, Bedeutung: Ende, es folgen keine Optionen mehr, der TCP-Header beginnt.

¹⁰<http://www.squid-cache.org/>

Nop: Typ: 1, keine Längenangabe, Bedeutung: No Operation, dieses ist ein Füllwort, welches keine Änderung bewirkt

MSS: Typ: 2, Len: 4 Die als MSS angekürzte Maximum Segment Size gibt an, wie viele Bytes lang das gesamte Datenpaket maximal werden darf.

Wscale: Typ: 3, Len: 3, Bedeutung: Um diesen Exponenten, der bei dem Verbindungsaufbau festgelegt wird, wird die Window Size vergrößert, um Werte von mehr als 65535 (16 Bit) darstellen zu können

SackOK: Typ: 4, Len: 2, Bedeutung: Der sendende Rechner bietet an, den in RFC2028 beschriebenen **Sequential ACK** Mechanismus zu unterstützen. Hierbei handelt es sich um ein Verfahren, bei dem eine Reihe von Paketen als fehlerfrei übertragen bestätigt werden können. Nun muss der Sender nur noch die restlichen Pakete erneut übertragen und nicht mehr alle Pakete dieser Serie.

Sack: Typ: 5, Len: variabel ,Bedeutung: Hier können bis zu vier Gruppen an Paketen aufgezählt werden, deren fehlerfreier Empfang bestätigt werden soll.

Time: Typ: 8, Len: 10, Bedeutung: Hier können zwei Zeitstempel übertragen werden. Diese werden als *timestamp value* und *timestamp echo reply* bezeichnet. Da nur das SYN-Paket von Interesse ist, ist der zweite Zeitstempel immer Null und braucht nicht ausgewertet zu werden.

Die Optionen nach RFC1644 ermöglichen *TCP Extensions for Transactions Functional Specification* (T/TCP). Diese Art der Verbindung ist ein Zwischending zwischen TCP und UDP, bei dem der Drei-Wege-Verbindungsaufbau abgekürzt wird. Viele Rechner unterstützen dieses Verfahren¹¹, aber kaum einer gibt diese Option bekannt. Da diese Option im Rahmen der Untersuchungen nur einmal auftauchte (die Mailserver von yahoo geben diese Option an), wird sie vorerst nicht ausgewertet.

¹¹FreeBSD beispielsweise unterstützt dieses Verfahren seit Version 2.0.5 (Juni 1995)

2. Problemanalyse

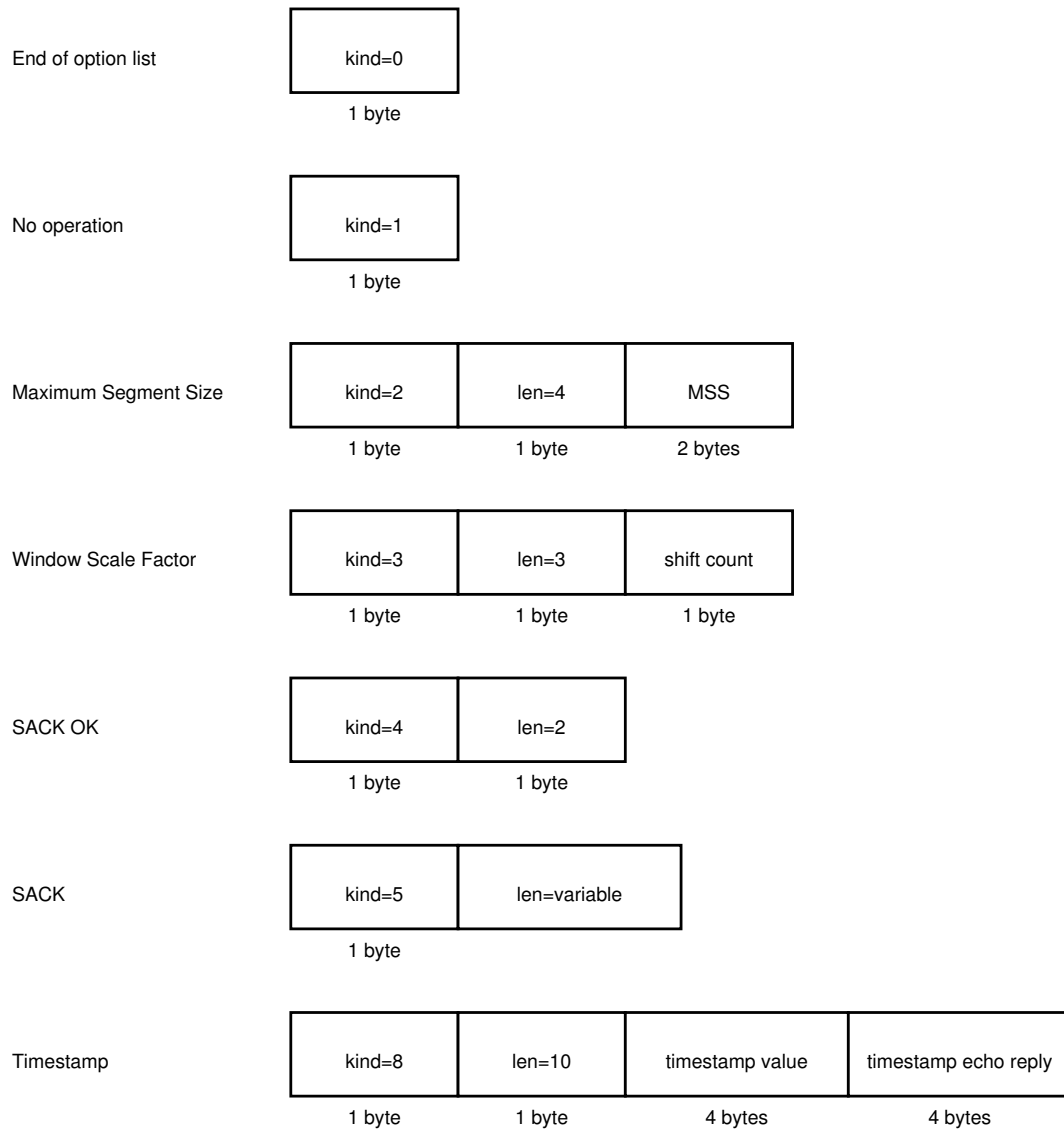


Abbildung 2.3.: TCP Optionen

2. Problemanalyse

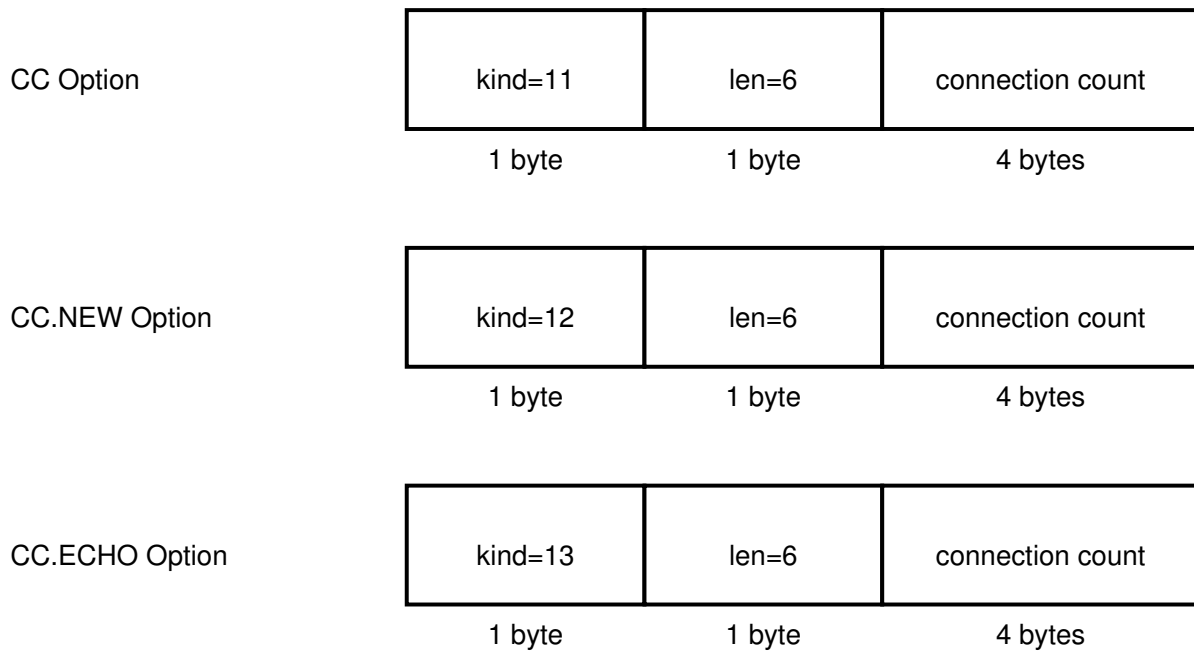


Abbildung 2.4.: TCP Optionen nach RFC1644

2.2.4. Application Layer

Meist sind diese Werte sehr einfach zu bekommen, da sie im Klartext übertragen und in die Logfiles geschrieben werden.

Beispiel aus der Logdatei eines *Apache-Webservers*:

```
80.131.137.31 - - [03/Jan/2004:16:26:16 +0100] \
  "GET /Berlin/tn/tn_DSC00077.JPG HTTP/1.1" \
  200 10162 "http://fuckner.net/Berlin/" \
  "Mozilla/5.0 (Windows; U; Windows NT 5.1; \
  de-DE; rv:1.4) Gecko/20030619 Netscape/7.1 (ax)"
```

Hier hat ein Nutzer am 3. Januar 2004 um 16:26 ein Windows XP (Windows NT 5.1) mit einem Mozilla 1.4 Browser, der deutsche Sprache bevorzugt, genutzt, um sich das Bild tn_DSC00077.JPG ausgeben zu lassen. Der Nutzer, der einen Zugang über T-online (80.131.137.31) hat, nutzt als

2. Problemanalyse

Übertragungsprotokoll HTTP/1.1 und der Referrer sagt, dass er sich vorher `http://fuckner.net/Berlin/` angeschaut hat.

Aber auch Server geben Versionsbezeichnungen aus, wie man in diesem Beispiel sehen kann:

```
$ telnet dedi.fuckner.net 22
Trying 81.169.152.140...
Connected to dedi.fuckner.net.
Escape character is '^]'.
SSH-2.0-OpenSSH_3.6.1p1 FreeBSD-20030924
```

Der Server meldet sich als SSH-Server, der das Programm OpenSSH in der Version 3.6.1p1 verwendet. Er akzeptiert SSH-Verbindungen, die das Protokoll SSH2 verwenden; andererseits wird er die Verbindung abbrechen (SSH-1.99 würde angeben, dass auch Protokollversion 1 akzeptiert wird, falls der Client Version 2 nicht unterstützt). Es ist der SSH-Server, der bei dem Basissystem von FreeBSD mitgeliefert wurde und die letzte Aktualisierung erfolgte am 24. September 2003.

Neben dem Auswerten der Logfiles und Versionsbezeichnungen ("Banner-scanning", kann man auch der von einem Programm versendeten Befehle an einen Server Eigenarten erkennen. Dieses Verfahren wird als *Application Fingerprinting* beschrieben.

3. Implementation

Wichtig für diese Aufgabe ist es, dass man zwar an die Daten der Kunden herankommt, diese aber davon selbst nichts mitbekommen. Deshalb wird hier ein passives Fingerprinting verwendet, welches selbst keine Pakete aussendet.

Dieses Programm soll die Netzwerkverbindungen zu dem entsprechenden Rechner überwachen. Dafür muss man alle Datenpakete, die im Netzwerk sichtbar sind, analysieren. Zu beachten ist, dass man bei Rechnern, die mit einem Hub verbunden sind, alle Datenpakete sehen kann, die die Rechner untereinander hin- und herschicken. Bei einem Switch hingegen sind nur die Pakete sichtbar, die für den eigenen Rechner bestimmt sind.

Ziel ist es, den Aufbau der Datenpakete im Netzwerk anzuschauen und anhand der Charakteristika der Datenpakete auf Betriebssysteme schliessen zu können.

Ferner ist es wichtig, das Projekt in mehrere Programme aufzuteilen; eines erhebt die Daten und ein anderes wertet diese aus. So kann man mehrere Rechner erfassen und später gemeinsam die Ergebnisse berechnen. Dies hat allerdings den Nachteil, dass akute Gefahren (wie Würmer), die eine neue Schwachstelle in Computersystemen ausnutzen, erst mit der meist monatlich ausgeführten Loganalyse stattfindet, erkannt werden. Dies ist zu spät. Es muss daher eine Zwischenanalyse stattfinden.

Die Entwicklung der Software erfolgte unter FreeBSD 5.2 und in der Firma incremental unter Suse Linux 9.0, welches garantiert, dass die Software auf beiden Systemen lauffähig ist. Die Anpassung an die Systeme übernehmen *autoconf* und *automake*. Eine Portierung auf andere Linuxsysteme ist nun trivial.

Die Programmierung erfolgt in der Sprache C, da diese nicht nur portabel ist, sondern auch extrem performant ist. Weiterhin existieren libraries wie die libpcap, welche Zugriff auf Hardwareebene ermöglichen. C-Compiler, wie der gcc¹ sind für viele Architekturen und Plattformen erhältlich. Der Kernighan/ Ritchie[3] half bei der Umsetzung.

Die Dokumentation wird mit L^AT_EX geschrieben.

¹<http://gcc.gnu.org/>

3.1. Pcap

Um den Datenstrom auf dem Netzwerkinterface komplett unbearbeitet (raw), mitlesen zu können, schaltet man das Netzwerkinterface in den *promiscuous mode*. Das bedeutet, dass das Netzwerkinterface **alle** Datenpakete annimmt, egal, ob die Pakete für den Rechner bestimmt sind, oder für einen anderen. Hier hilft die *libpcap*, eine Programmierschnittstelle zum Auslesen von Daten aus dem Netzwerk (*pcap* steht für **p**acket **c**apture). Sie ist Bestandteil des Programmes *tcpdump* und ermöglicht es, den Rohdatenstrom am Netzwerkinterface auszulesen. Hierbei ist zu beachten, dass *libpcap* die Datenpakete **vor** einer eventuell installierten Firewall bekommt, weil alle Pakete ausgewertet werden sollen und nicht nur die Anfragen, die von den Firewallregeln erlaubt sind. Die Daten werden direkt am *BPF*, dem *Berkeley Packet Filter* entnommen. Diese Schnittstelle erlaubt es, protokollunabhängig Rohdaten auf ISO/OSI-Ebene 2 abzugreifen. Wie der Name schon sagt, entstammt diese Schnittstelle den BSD-Unixen. Da es sich bei der Ausgabe um einen Rohdatenstrom handelt, muss der Offset ermittelt werden, der aussagt, wo in diesem Datenstrom der IP, beziehungsweise TCP-Header beginnt. Das Programm erkennt die Offsets für PPP (Modem, ISDN, 4 Bytes), PPPoE (DSL-Verbindungen, 8 Bytes beziehungsweise 16 Bytes unter Linux) und Ethernet (14 Bytes) und deckt damit alle relevanten Netzwerkverbindungen ab. Zur Bestimmung des Offsets schaut man sich einfach an, an welcher Stelle im Datenstrom die für den Beginn des IP-Headers charakteristischen 45 stehen. Dies ist schon in [1.2](#) behandelt.

Dieses Programm benötigt die vollen Benutzerrechte. Unter UNIX-ähnlichen Betriebssystemen ist das der Benutzer *root* mit der User-ID (uid) Null. Da ein Programm mit diesen Rechten eine potenzielle Sicherheitslücke darstellt, wird das Programm mit folgenden Befehlen befähigt, mit root-Rechten ausgeführt zu werden:

```
chown root pcaptest
chmod +s pcaptest
```

Danach sieht man das gesetzte suid-Bit mit dem Befehl `ls -l`

```
-rwsr-sr-x  1 root          wheel      61745 Jan 30 15:49 pcap
```

Nun kann ein normaler Benutzer dieses Programm ausführen und dieses Programm wird mit root-Rechten gestartet. Das Verfahren hat den Vorteil, dass der Nutzer nur dieses eine Programm ausführen kann und nicht das Administratorkennwort ausgehändigt bekommen muss.

3. Implementation

Da das Programm nur zum Öffnen der Netzwerkschnittstelle diese erhöhten Rechte benötigt, werden sofort nach dem Durchlauf dieser Programmstellen die erhöhten Rechte wieder abgegeben. Dies geschieht folgendermaßen:

```
if (geteuid() != getuid())
    setuid(getuid());
if (geteuid() != getuid())
    fprintf(stderr, "cannot drop privileges");
```

Es wird hier geprüft, ob die User-ID (uid) und die effektive User-ID (euid) übereinstimmen. Ist dies nicht der Fall (also wurde ein Programm mit gesetztem suid-Bit aufgerufen, ist hier die eid Null, während die uid der User-ID des Nutzers entspricht. Es wird danach versucht, dem Programm die Rechte des echten Users zu geben. Danach wird der selbe Sachverhalt noch einmal geprüft. Sind uid und eid immer noch nicht identisch, wird eine Fehlermeldung ausgegeben.

Nun kann das Programm das Netzwerkinterface öffnen und daraus lesen. Hier ist es wichtig, dass nicht alle Datenpakete analysiert werden, sondern, dass man vorher überlegt, welche Pakete die relevanten Daten enthalten.

Damit nicht jedes Paket von der Netzwerkschnittstelle aufgenommen wird, gibt es eine eigene Sprache, mit der man einen Filter definieren kann.

```
pcap_filter: src not 81.169.152.140 and
    (icmp or (tcp[13] & 2 != 0 and not tcp[13] & 16 != 0))
```

Diese Filterregel sagt aus, dass die Pakete durch den Filter kommen, die nicht von der lokalen IP-Adresse stammen. Dies wäre auch sinnlos, da man den eigenen Rechner nicht zu analysieren braucht. Dann akzeptiert der Filter alle Pakete des Typs ICMP (für zukünftige Erweiterungen) oder TCP-Pakete, die das SYN-Flag *tcp[13]* (2. Bit) und nicht das ACK-Flag *tcp[13]* (4. Bit) gesetzt haben. Das bedeutet, dass eingehende Verbindungsanfragen analysiert werden und nicht deren Antwort, bei dem beide dieser Flags gesetzt sind.

Der mit *pcap_filter* bearbeitete String wird ausgewertet und direkt auf den *BPF* angewendet. Diese Arbeit übernehmen vorhandene Systemcalls.

Das Programm besitzt eine Endlosschleife, den *network_loop*, welcher die Funktion *network_analyse* aufruft, wenn ein Paket empfangen wurde, das den Kriterien entspricht. In dieser Funktion wird durch Angabe des Offsets der Datenstrom eingeteilt in die Header der

3. Implementation

verschiedenen Protokolle/ Layer. Die einzelnen Felder des Headers werden im Folgenden analysiert. Wenn das Programm mit der Option verbose (-v) aufgerufen wurde, werden all diese Informationen auf die Standardausgabe geschrieben. In jedem Fall werden aber die Daten in einem String gesammelt und nach der Analyse in die Ausgabedatei geschrieben.

Das Programm gibt im Normalfall eine Zeile mit Charakteristika aus, die auch in die Textdatei geschrieben wird. Als Format eignet sich eine Textdatei, da diese besonders einfach wieder einzulesen ist. Üblicherweise sind die von Serverprogrammen geschriebenen Logfiles auch in diesem Format. Als Trennzeichen wird das Semikolon verwendet. Andere Formate wie beispielsweise die *Extensible Markup Language*² (XML) können auch verwendet werden, doch ist es aufwendiger, diese wieder einzulesen.

```
$ ./pcap -v -i fxp0
pcap_filter: src not 81.169.152.140 and
  (icmp or (tcp[13] & 2 != 0 and not tcp[13] & 16 != 0))
Interface: fxp0      snaplen: 100      promisc: 1 buffering: 1
00 30 48 52 93 ac 00 0b fc b1 ef 00 08 00      Ethernet

IP Header
-----
45 00 00 3c      IPv4, IPlen: 5, TOS: 0 length: 60
29 e0 40 00      ID: 10720, Flags: RF0 DF1 MF0 frag offset: 0
38 06 06 b3      TTL: 56 Protocol: 6 Checksum: 1715
d5 b2 52 41      Source: 213.178.82.65
51 a9 98 8c      Destination: 81.169.152.140

TCP Header
-----
e7 1f 00 50      Src Port: 59167, Dst Port: 80
24 77 61 0b      Sequence Number: 611803403
00 00 00 00      Acknowledgement Number: 0
a0 02 ff ff      Len: 10 Flags: SYN WSize: 65535
82 d9 00 00      TCP Checksum: 55682 Urgent Pointer: 0
02 04 05 b4
01 03 03 01
01 01 08 0a
02 07 47 0b
00 00 00 00
```

²<http://www.w3.org/XML/>

3. Implementation

Wert	Bedeutung
1073128651	Timestamp in UNIX-Zeit
213.178.82.60	IP-Adresse des anfragenden Rechners
60	IP-Header-Länge
10720	IPID
1	DF-Bit ist gesetzt
64	TTL (geschätzt)
6	Protokoll (TCP)
###	TCP-Header beginnt
59167->80	Quellport -> Zielport
SYN	Typ
65535	WindowSize
0	urgent pointer
1460	mss
1	window scale
MNWNNT	Optionen: MSS, Nop, WScale, Nop, Nop, Time
96163722	timestamp a

Tabelle 3.1.: Relevante Partien des TCP und IP-Headers

output:

```
1073128651;213.178.82.65;60;10720;1;64;6### \  
59167->80;SYN;65535;0;1460;1;MNWNNT;96163722
```

Die ersten zwei Zeilen werden nur beim Start ausgegeben und geben an, mit welchen Optionen das Programm arbeitet. Danach folgen die Header des Host-To-Network-Layer, also die Ziel- und die Quellmacadresse. Es handelt sich hier um Ethernet, wie an dem folgenden *08 00* zu erkennen ist.

In der letzten Zeile stehen die relevanten Informationen, die in die Ausgabedatei geschrieben werden. Die Daten setzen sich folgendermaßen zusammen:

Ohne das verbose-Flag wird nur diese Zeile in der Konsole ausgegeben.

3.2. Parser - Monatliche Auswertung

Die vom Programm *pcap* gewonnenen Daten müssen natürlich ausgewertet werden. Dieses kann auf einem anderen Rechner geschehen. Auch können mehrere Server gemeinsam analysiert werden, wenn man die einzelnen Logfiles einfach verkettet. Dieses Vorgehen macht Sinn, wenn mehrere Computersysteme die selbe Aufgabe haben und die Lastverteilung durch einen Load-Balancer vorgenommen wird. Da diese Computer wie ein einzelner Computer auftreten, ist es ratsam, die Logfiles dieser Rechner gemeinsam zu analysieren.

Zu diesem Zweck gibt es das Programm *parse*, welches folgende Aufgaben hat:

- Einlesen der Ausgabe des PCAP-Programmes (PCRE)
- Einlesen der Referenzdaten *fingerprints*
- Vergleich der gewonnenen Daten mit den Referenzdaten
- graphische Aufbereitung der Ergebnisse (HTML,GD)

Das Einlesen der von *pcap* geschriebenen Informationen kann auf verschiedenen Wegen geschehen. Man könnte die Datei von Hand parsen, also selbst die Datei zeichen für Zeichen zerlegen. Dieses Verfahren kann schnell ausgeführt werden, aber die Programmierung ist zeitaufwendig.

Eine weitere Möglichkeit wäre die Verwendung von *flex*³ in Verbindung mit *bison*⁴. *Flex* ist ein Werkzeug zum Erzeugen von Programmen, die Textdateien nach Mustern untersuchen. *Bison* erzeugt aus vorgegebenen Regeln C-Programme, welche Textdateien entsprechend analysieren.

Der *langsamste* Weg sind reguläre Ausdrücke. Hier kann man mit komfortablen Regeln den Aufbau der Ausgabedatei von *pcap* und der Referenzdaten *fingerprints* einlesen. Dieser Geschwindigkeitsnachteil ist zwar meßbar, aber da die Auswertung nur monatlich stattfindet, ist dies zu verschmerzen.

Letztendlich fiel die Wahl auf *Perl Compatible Regular Expressions*⁵ (PCRE). *Pcre* hat den Vorteil, dass man die komplexe Syntax von *perl* in einem C-Programm nutzen kann. Mit *pcre* definiert man den Aufbau der *fingerprints/ outputfile*-Dateien:

³<http://www.gnu.org/software/flex/>

⁴<http://www.gnu.org/software/bison/>

⁵<http://www.pcre.org/>

3. Implementation

```
fpd.fprints= pcre_compile("^[0-9]{1,5});" // 1 - ipheaderlen
"([0-9]{1,5});" // 2 - ip_id
"([01]);" // 3 - ip_df
"([0-9]{1,3});" // 4 - ip_ttl
"([0-9]{1,3})" // 5 - ip_proto
"###" //tcpheader beginnt
"([0-9]{1,5});" // 6 - window size
"([0-9]{1,5});" // 7 - urg
"([0-9]{1,4});" // 8 - mss
"([0-9]);" // 9 - scale
"([A-Z]{0,19});" // 10 - options
"([0-9]{1,15});" // 11 - timestamp a
"(.{1,49})" // 12 - Description
"$"
,0
,&error
,&erroffset
,NULL);
```

Dieser Filter zerlegt den Inhalt der fingerprints-Datei, der Filter für die Ausgabedatei *outputfile* sieht aber sehr ähnlich aus. Die Anweisung beginnt mit dem zu suchenden Muster. Für die IP-Headerlänge beispielsweise sind die Ziffern null bis neun erlaubt, ihre Anzahl muß eine bis fünf betragen. Bei der IPID sind die Ziffern null oder eins zugelassen, da keine Anzahl angegeben ist, erwartet pcre genau ein Zeichen. Die Beschreibung beschreibt ein bis 49 Zeichen; der Punkt symbolisiert beliebigen Inhalt. Weiterhin sind folgende Optionen angegeben: das Dollarzeichen steht für das Zeilenende. Alles, was nach dem 49. Zeichen bis zum Zeilenende folgt, wird ignoriert. Danach kann man bei den folgenden Optionen z.B. auswählen, dass Groß- und Kleinschreibung ignoriert wird. Diese Optionen werden nicht genutzt. Tritt ein Fehler auf, wird eine Fehlermeldung an die Adresse von *error* geschrieben, an die Adresse von *erroffset* wird die Nummer des Musters geschrieben, in dem der Fehler auftrat. Außerdem kann man noch *locales*, also unterstützte Schriftzeichensätze angeben. Da diese Option nicht benötigt wird, wird NULL als Wert angegeben.

Sind die Fingerprints eingelesen, liest das Programm *parse* zeilenweise die Ausgabedatei *outputfile* ein und vergleicht diese Elemente mit den Fingerprints. Ist ein Betriebssystem ermittelt, wird dieser Wert gespeichert.

3. Implementation

Nach der eigentlichen Auswertung folgt die graphische Aufbereitung mit `gd`⁶, da man mit diesem Programm relativ einfach Tortengrafiken erzeugen kann. Außerdem bietet die Bibliothek `libgd` Schnittstellen zu C und PHP, was die Programmierung einfach macht. Alternativ gibt es `gnuplot`⁷, welches allerdings eher für normale Graphen, besonders dreidimensionale Graphen, geeignet ist.

Für das Erstellen der Tortengrafiken mit `gd` benötigt man folgende Befehle:

```
colours[13] = gdImageColorAllocate(im, 0, 0, 0);           //black
temp2=temp=270;

gdImageFilledRectangle(im,0,0,500,600,colours[14]);

for (i=0;i<10;i++) {
    bzero(servicename, sizeof(servicename));
    temp=temp2;
    temp2+=(int)(360*ports[i].count/values);
    for (j=0;servicenames[j].service != NULL;j++)
        if (servicenames[j].port==ports[i].pnum)
            break;
    if (servicenames[j].service != NULL)
        sprintf(servicename,"%s (%d, %.2f%%)",servicenames[j].service,
            ports[i].pnum,100*(ports[i].count/values));
    else
        sprintf(servicename,"unknown (%d, %.2f%%)",ports[i].pnum,
            100*(ports[i].count/values));
    rest-=100*(ports[i].count/values);
    gdImageFilledRectangle(im,20,50*i+30,40,50*(i+1),colours[i]);
    err=gdImageStringFT(im,&brect[0],colours[12],f,sz,0.0,50,50*(i+1),
        servicename);
    gdImageFilledArc(im,350,150,200,200,temp,temp2,colours[i],gdArc);
}
bzero(servicename, sizeof(servicename));
sprintf(servicename,"Rest (%.2f%%)",rest);
gdImageFilledRectangle(im,20,50*i+30,40,50*(i+1),colours[i]);
err = gdImageStringFT(im,&brect[0],colours[12],f,sz,0.0,50,50*(i+1),
    servicename);
```

⁶<http://www.boutell.com/gd/>

⁷<http://sourceforge.net/projects/gnuplot/>

3. Implementation

```
gdImageFilledArc(im, 350, 150, 200, 200, temp2, 270, colours[i], gdArc);
pngout = fopen("ports.png", "wb");
gdImagePng(im, pngout);
fclose(pngout);
gdImageDestroy(im);
```

Die einzelnen Befehle bewirken folgendes:

`gdImageCreate` : definiert die Fläche, die beschrieben werden kann, hier 500x600 Pixel

`gdImageColorAllocate` : definiert Farben in RGB-Notation, hier rot

`temp2=temp=270` : Pointer beginnen bei 270 Grad (oben)

`gdImageFilledRectangle`: füllt den Hintergrund mit Farbe, bzw. die Felder vor den Beschreibungen

`dImageFilledArc` : zeichnet ein Tortenstück von temp bis temp2.

`gdImageStringFT` : Schreibt einen String an eine angegebene Position in das Bild

`pngout = fopen(ports.png", "wb")`: Öffnen der Ausgabedatei, Ausgabeformat ist PNG.

`gdImagePng(im, pngout)`: Schreibt das erstellte Bild in die Ausgabedatei

`fclose(pngout)` : Schließt die Ausgabedatei

`gdImageDestroy(im)` : Entfernt das Bild aus dem Speicher

Als Ausgabeformat der Grafiken wurde PNG⁸ gewählt, da dieses Format auf allen Architekturen/ in allen Browsern darstellbar ist und ohne rechtliche Einschränkungen genutzt werden darf (im Gegensatz zu JPG oder GIF). Das Patent der Firma Unisys⁹ auf das GIF-Grafikformat in den USA am 20. Juni 2003 ab, aber in vielen anderen Ländern ist es noch bis zum 7. Juli 2004 gültig. Aufgrund dieser Tatsache wurde die Unterstützung der GIF-Formates aus der aktuellen Version von *gd* (2.0.21) entfernt.

Grafiken werden in PNG **verlustfrei** komprimiert und die erzeugten Dateien sind bei einer geringen Anzahl von Farben wesentlich kleiner als JPG, GIF oder BMP. Man kann in PNGs Schriften einbinden, idealerweise benutzt man True Type Fonts (TTF¹⁰), da diese skalierbar

⁸<http://www.w3.org/Graphics/PNG/>

⁹<http://www.unisys.com/>

¹⁰<http://www.trueType.demon.co.uk/>

sind, also nicht ausfransen, wenn man ihre Größe ändert. Die Wahl fiel auf die Schriftart Tahoma, weil diese sehr gut lesbar ist. Obwohl diese auf den meisten Systemen installiert ist, ist sie keine Freeware, sondern mit dem Copyright von Microsoft versehen.

Die Schrift ist als Variable in der Datei *defines.h* eingetragen und kann somit leicht ausgetauscht werden.

Im Internet gibt es eine Vielzahl kostenpflichtiger aber auch freier TTFs. Eine gute Übersicht findet sich auf der Seite des Niederländers Martijn Katwijk¹¹.

3.3. Parser - Echtzeitauswertung

Permanent an der Internet angeschlossene Rechner sind zu jedem Zeitpunkt einem Risiko ausgesetzt, wenn eine neue Sicherheitslücke bekannt wird oder sich ein neuer Wurm verbreitet. Da die Bandbreite der an das Internet angeschlossenen Rechner kontinuierlich zunimmt, breiten sich auch diese Gefahren immer schneller aus.

Da eine reguläre Analyse von Logfiles generell monatlich stattfindet, ist dies nicht ausreichend, da ein Rechner, der so eine Sicherheitslücke aufweist, schon längst infiziert ist. An dieser Stelle ist anzumerken, dass dieses Programm in keiner Weise Infektionen durch Viren oder Würmer verhindern kann oder soll, es soll lediglich Anfragen auswerten und Statistiken erstellen.

Einen weiteren sicherheitstechnischen Aspekt hat dieses Programm dennoch, allerdings kommt dieser erst zum tragen, wenn es eigentlich schon zu spät ist. Sollte es jemals passieren, dass durch eine Schwachstelle in den eingesetzten Programmen es einem Eindringling ermöglicht wird, sich Zugang zu verschaffen, wird dieses weiterhin protokolliert. Üblicherweise ersetzen diese *Rootkits* genannten Programme einige Systemdienste, um unerkannt zu bleiben. Diese neuen Programmdateien verstecken die nun eingebauten Hintereingänge, die es ermöglichen, sich jederzeit wieder auf dem Computersystem anzumelden. Manche dieser Rootkits schalten das Netzwerkinterface in den promiscuous mode, um eingehende Netzwerkverbindungen und Klartextpasswörter mitlesen zu können. In diesem Falle können eingehende Netzwerkverbindungen nicht maskiert werden, da dieses Programm das Netzwerkinterface, genauer gesagt den BPF schon belegt, solange es nicht beendet wird.

Zu diesen Zweck wird alle fünf Minuten eine Zwischenauswertung gemacht. Hier ist es ausreichend, die angefragten Ports zu ermitteln. Dafür wird das Parse-Programm stark vereinfacht,

¹¹<http://www.aaa.nl/people/mkatwijk/freettf.html>

3. Implementation

so dass dort nur die angefragten Ports ermittelt werden. Die graphische Aufbereitung erfolgt nicht mit `gd`, sondern `rrdtool`. Der Vorteil dieses Programmes ist, dass es bei zeitabhängigen Werten diese in einer Datenbank archiviert und im Laufe der Zeit Mittelwerte bildet. Dieses Verfahren hat den Vorteil, dass bei Daten aus der Vergangenheit nicht mehr alle Werte benötigt werden, es reicht meist aus, Ereignissen, die ein halbes Jahr her sind, zu erkennen, aber die genauen Werte sind dann uninteressant. Dieses Zwischenauswertungsprogramm wird über ein Skript gestartet, welches alle fünf Minuten von `cron`¹² aufgerufen wird. Das Skript sieht folgendermaßen aus:

```
$ cat rotate.sh
#!/bin/sh
cd /home/molli123/BA/src
mv outputfile outputfile.curr
killall -SIGHUP pcap
./rrd
cat outputfile.curr >>outputfile.full
./graph.sh
```

Dieses Skript rotiert die Logfiles alle 5 Minuten, informiert `pcap`, dass der pointer auf die Ausgabedatei neu geöffnet werden muss und ruft das vereinfachte `parse` auf, welches die Daten in einer RRD-Datenbank ablegt. Danach wird die gerade analysierte Datei an die Gesamtlogdatei angehängt. Diese wird dann der monatlichen Analyse zugeführt.

Die RRD-Datenbank wird einmal angelegt und kümmert sich selbständig um das Verwalten der Daten und das Erzeugen der Grafiken. Das Anlegen erfolgt so:

```
$ cat create.sh
rrdtool create ports.rrd \
  --step 300 \
  DS:http:GAUGE:600:0:100 \
  DS:loc-srv:GAUGE:600:0:100 \
  DS:microsoft-ds:GAUGE:600:0:100 \
  DS:rest:GAUGE:600:0:100 \
  RRA:AVERAGE:0.2:1:288 \
  RRA:AVERAGE:0.5:288:31
```

¹²`cron` ist ein UNIX-Systemdienst, der periodisch Aufgaben ausführt

3. Implementation

Es wird eine Datei *ports.rrd* angelegt, welche alle fünf Minuten (step 300) einen neuen Wert erwartet. *GAUGE* bedeutet, dass es sich hier um Einzelwerte handelt. Wenn nicht alle zehn Minuten (600 Sekunden) ein neuer Wert eingeht, wird dieser als UNKNOWN angenommen und in der Gesamtauswertung interpoliert. Die Werte müssen zwischen 0 und 100 liegen.

Graphen erzeugt man mit

```
$ cat graph.sh
/usr/local/bin/rrdtool graph portsrrd.png -a PNG --title="Requested Ports" \
  --height 150 --vertical-label="Requested Ports" \
  DEF:http=ports.rrd:http:AVERAGE \
  DEF:loc=ports.rrd:loc\-srv:AVERAGE \
  DEF:microfs=ports.rrd:microsoft\-ds:AVERAGE \
  DEF:rest=ports.rrd:rest:AVERAGE \
  AREA:rest#ff0091:"Rest" \
  STACK:loc#00b871:"LOC-SRV (135)" \
  STACK:microfs#d9dc26:"MICROSOFT-DS (445)" \
  STACK:http#0022e9:"HTTP (80)" \
  COMMENT:"\s" \
  COMMENT:"\s" \
  COMMENT:"Graph created: $(date +%Y-%m-%d\ %H:%M)"
```

3.4. Ausgabe

So sieht die fertige Ausgabe der beiden Parser aus, wenn mittels rrdtool bzw. gd Grafiken erzeugt wurden. Das Programm kann etwa 30000 Einträge pro Sekunde auswerten.

3. Implementation

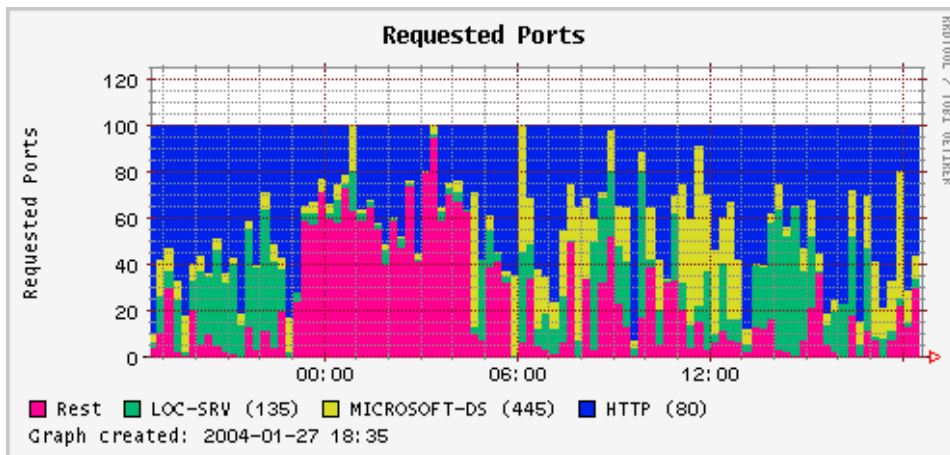


Abbildung 3.1.: Echtzeitauswertung angefragter Ports

3. Implementation

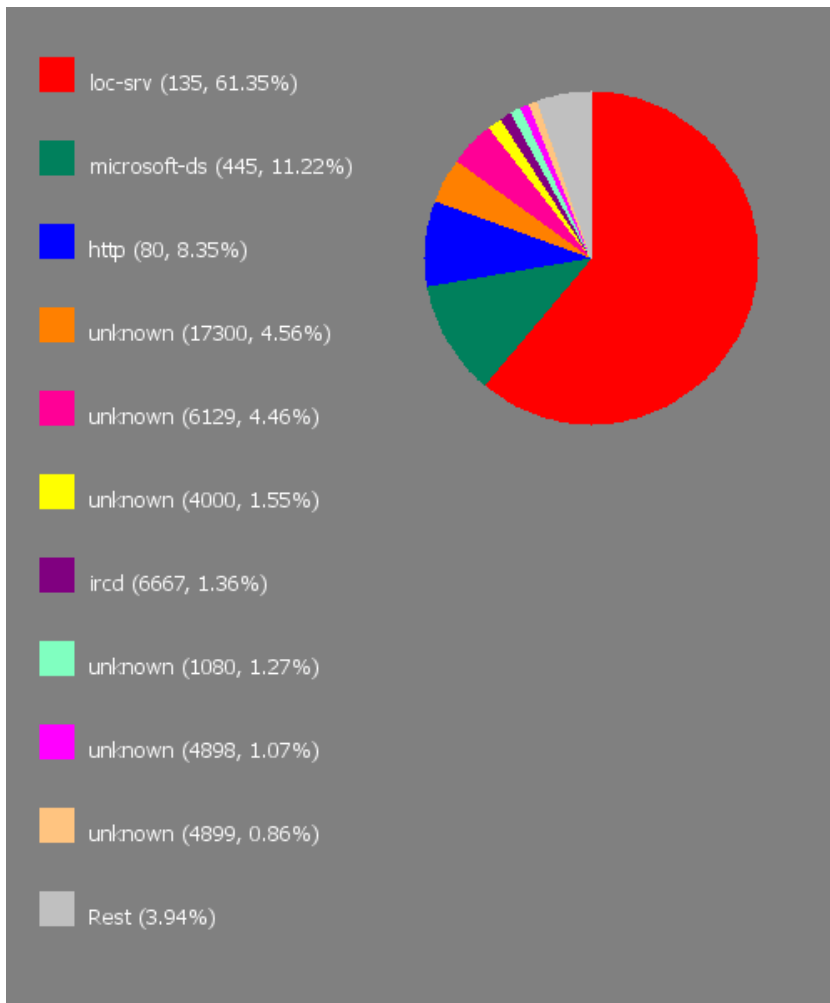


Abbildung 3.2.: Gesamtanalyse angefragter Ports

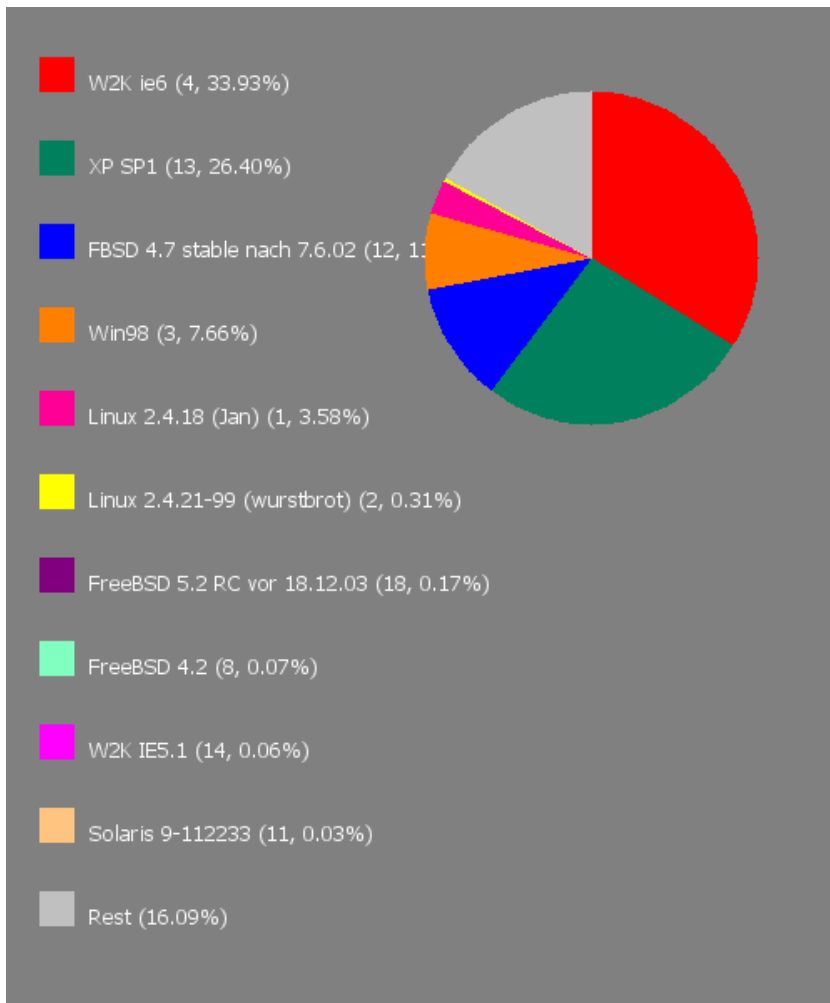


Abbildung 3.3.: Auswertung anfragender Betriebssysteme

3.5. Installation

Das Programm wird, wie in der UNIX-Welt üblich, als *tgz*¹³-Datei bereitgestellt. Diese wird mit dem Befehl `tar xvzf pcap.tgz` ausgepackt. Nun ruft der Nutzer das Skript `autogen.sh` auf, welches die Programme `autoheader`, `aclocal`, `automake` und `autoconf` der Reihe nach aufruft.

¹³entspricht `tar.gz` und ist ein mit `gzip` komprimiertes `tar`-Archiv

Autoheader ist Bestandteil des *autoconf*-Paketes und *aclocal* ist im *automake*-Paket enthalten. Anschliessend wird das dabei erstellte *configure*-skript ausgeführt.

```
$ cat autogen.sh
#!/bin/sh
${AUTOHEADER}
${ACLOCAL}
${AUTOMAKE} -a
${AUTOCONF}
./configure
```

Mit dem Befehl *make* werden die Quelldateien dann kompiliert und zu ausführbaren Binärdateien gelinkt.

Nun gilt es noch, die Programmdatei *pcap* mit den erhöhten Nutzerrechten auszustatten. Zu diesem Zweck existiert im Verzeichnis *src* das Skript *suid.sh*. Zum Ausführen dieses Skriptes muss man einmalig als root angemeldet sein, um weitere Rechte vergeben zu dürfen.

Zur regelmäßigen Analyse wird das Skript alle fünf Minuten vom Systemdienst *cron* aufgerufen; die dort erhobenen Daten werden in die RRD-Datenbank gespeichert.

```
* /5 * * * * /home/molli123/BA/src/rotate.sh
```

3.6. Verwendete Software

autoconf : *Autoconf* ist ein Paket von m4-Makros, welche Shell-Skripte zum automatischen Konfigurieren von Quellcodepaketen erzeugen. Diese Skripte können Pakete an viele UNIX-ähnliche Umgebungen ohne Eingreifen des Nutzers anpassen. *Autoconf* erzeugt ein Konfigurationsskript aus einer Vorlage, welche angibt, welche betriebssystemspezifischen Merkmale das Programm nutzen kann.

Verwendete Version: 2.57

<http://www.gnu.org/software/autoconf/autoconf.html>

automake: *Automake* ist ein Programm, welches dazu dient, *Makefile.in*-Dateien zu erzeugen, die den GNU Programmierungs Standards entsprechen.

Verwendete Version: 1.7

<http://www.gnu.org/software/automake/automake.html>

3. Implementation

- cvs** : *CVS* steht für *Concurrent Versions System*, das führende open-source, netzwerkfähige Versionskontrollsystem. CVS ist nützlich für jeden Programmierer, von einzelnen Entwicklern bis hin zu grossen, verteilten Gruppen. CVS verleiht Entwicklern Zugang zur aktuellen Quellcodeversion, einzige Voraussetzung ist ein Internetzugang. Es löst Versionskonflikte und ist für die meisten Plattformen erhältlich. Ein Versionsmanagementprogramm ist unerlässlich, wenn man Programmdateien unter verschiedenen Betriebssystemen an verschiedenen Standorten bearbeiten will. Hilfreich ist das Buch *Open Source-Projekte mit CVS* [4].
Verwendete Version: 1.11.5-FreeBSD
<http://www.cvshome.org>
- dia** : *Dia* ist ein an Visio¹⁴ angelehntes Programm zum Erstellen von UML-, Fluss- und Netzwerkdiagrammen, welches einfach um neue Symbole erweitert werden kann.
Verwendete Version: 0.92.1 [http://www.lysator.liu.se/ alla/dia/](http://www.lysator.liu.se/alla/dia/)
- gcc** : *Gcc* steht für *GNU Compiler Collection*. Diese Sammlung an Programmen unterstützt C, C++, Objective-C, Fortran, Java, und Ada. Der *gcc* ist der Standard-C-Compiler, der auf bei den meisten Linuxdistributionen schon installiert ist.
Verwendete Version: 3.3.3
<http://gcc.gnu.org/>
- gd** : *Gd* steht für *Graphics-Draw Module* und ist eine ANSI-C-Grafikbibliothek zum schnellen Erzeugen von PNG und JPG-Grafikdateien. Die aktuelle Version (2.0.21) kann keine GIF-Dateien erstellen.
Verwendete Version: 2.0.15
<http://www.boutell.com/gd/>
- LaTeX** : \LaTeX ist eine Sammlung nützlicher Makros für das hochwertige Textsatzsystem \TeX . Dieses wurde für technische und wissenschaftliche Dokumentationen entwickelt und ist der *de facto* Standard für die Kommunikation und Publikation wissenschaftlicher Arbeiten. Als hilfreich für das Arbeiten mit \LaTeX erwies sich das Buch von Leslie Lamport.[5]
Verwendete Version: 7.5.4
<http://www.latex-project.org/>
- NEdit** : *NEdit* ist ein X11/Motif Texteditor, der neben \LaTeX und C viele andere Sprachen erkennt und mit korrektem Syntag-Highlighting versieht. Verwendete Version: 5.3
<http://www.nedit.org/>

¹⁴<http://office.microsoft.com/visio/>

3. Implementation

- pcre** : Die *pcre*-Bibliothek ist eine Sammlung von Funktionen, die reguläre Ausdrücke ermöglichen, die die selbe Syntax wie Perl 5 verwenden. *Pcre* besitzt sowohl eine eigene Programmierschnittstelle, als auch Funktionen, die den POSIX regulären Ausdrücken entsprechen.
Verwendete Version: 4.5
<http://www.pcre.org/>
- rrdtool** : RRDTool ist ein Programm zum Speichern und Darstellen zeitabhängiger Daten wie Netzwerkbandbreite, Raumtemperatur oder Server-Last. Es verwendet eine kompakte Art der Daten-Speicherung, welche in ihrer Größe konstant bleibt und bietet einfache Methoden, die gespeicherten Daten graphisch darzustellen.
Verwendete Version: 1.0.45
<http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/>
- Tcpdump**: *Tcpdump* ist ein Programm, welches ein low Level Interface zur Aufnahme von Paketen bereitstellt. Zu diesem Programm gehört *libpcap*, welches anderen C-Programmen ermöglicht, Pakete direkt von der Netzwerkschnittstelle aufzunehmen und selber weiterzuverarbeiten. Mithilfe der systemeigenen Schnittstellen und Headerdateien kann man auf den Inhalt der Pakete zugreifen.
Verwendete Version: 3.7.2 (tcpdump)/ 0.7 (libpcap)
<http://www.tcpdump.org>

Ausblick

Es wurde ein Programmgerüst geschaffen , weches aus den Protokollen TCP und IP implementationspezifische Eigenarten herausgreifen, analysieren und auswerten kann.

Eine Analyse mehrerer Pakete und die daraus resultierenden Informationen anhand der IPID und des Timestamps, wie auch die Auswertung der Protokolle ICMP und UDP wird als Nächstes folgen.

Für die Zukunft ist geplant, die Auswertung auf Ebene der MAC-Adressen und auf Applikationsebene (Banner und Eigenarten der Clientprogramme) zu erweitern und all diese in Relation zu setzen. Dort wird dann eine Prüfung auf Plausibilität zwingend notwendig. Empfehlenswert wäre ein System, welches Punkte für einzelne Charakteristika vergibt und so die möglichen Deutungen gegeneinander abwägt. Ferner ist eine Abschätzung auf unmögliche Relationen wie den Browser Internet Explorer und Linux zu prüfen und als falsch zu verwerfen, beziehungsweise auf ein NAT zu schliessen und selbständig zu versuchen, die dahinterliegenden Rechner zu analysieren.

Literaturverzeichnis

- [1] Andrew S. Tanenbaum. *Computer Networks, Third Edition*, Prentice Hall, 1996, ISBN 0-130-66102-3
- [2] W. Richard Stevens *TCP/IP Illustrated, Volume 1*, Addison-Wesley Publishing Company, 1996, ISBN 0-201-63346-9
- [3] Brian W. Kernighan, Dennis M. Ritchie *Programmieren in C, Zweite Ausgabe*, Hanser Fachbuchverlag, 1990, ISBN 3-446-15497-3
- [4] Karl Fogel, Bar Moshe *Open Source-Projekte mit CVS*, MITP, 2002, ISBN 3-82660816-X
- [5] Leslie Lamport *L^AT_EX User's Guide & Reference Manual*, Addison-Wesley Publishing Company, 1994, ISBN 0-201-52983-1

A. Glossar

- Autoconf** *Autoconf* erzeugt Shell-Skripte zum automatischen Konfigurieren von Quellcodepaketen, Seite 52
- Automake** *Automake* erzeugt *Makefile.in*-Dateien, die den GNU Programmierungsstandards entsprechen, Seite 52
- BPF** Der *Berkeley Packet Filter* ist eine Schnittstelle, die protokollunabhängig Rohdaten auf *Data Link*-Ebene abgreifen kann, Seite 38
- CVS** Programm zum Versionsmanagement von Quellcodedateien, Seite 53
- Dia** *Dia* ist ein an Visio angelehntes Programm zum Erstellen von Diagrammen, Seite 53
- GCC** Der Gnu C-Compiler erzeugt ausführbare Binärdateien aus Quellcode, Seite 37
- Gd** Programmierschnittstelle zum Erstellen von Grafiken im PNG oder JPG-Format, Seite 53
- Handshake** Der dreistufige Verbindungsaufbau bei TCP/IP wird als *Handshake* bezeichnet., Seite 22
- IANA** Die *Internet Assigned Numbers Authority* vergibt Nummern im Bereich des Internet, Seite 25

- IEEE Das *Institute of Electrical and Electronics Engineers* beschäftigt sich mit der Normung elektrotechnischer Standards, Seite 25
- IP Das *Internet Protocol* dient der Zustellung von Datenpaketen im Netzwerk, Seite 8
- ISO/OSI Das gebräuchliche Modell zur Darstellung der Kommunikation in Computernetzwerken, Seite 12
- LaTeX ist ein professionelles Textsatzsystem für technische und wissenschaftliche Dokumentationen, Seite 37
- MAC Die *Media Access Control* ist eine weltweit eindeutige Nummer, die eine Netzwerkkarte identifiziert, Seite 16
- MSS Die *Maximum Segment Size* gibt an, wie viele Bytes in ein Paket passen, ohne, dass die Daten fragmentiert werden müssen, Seite 21
- NAT Als *Network Address Translation* bezeichnet man eine Netzwerktopologie, bei der mehrere Computer auf eine externe IP abgebildet werden, Seite 16
- NEdit Graphischer Texteditor, der Syntaxhighlighting für \LaTeX , C und viele andere Sprachen darstellt, Seite 53
- PCRE Die *pcre*-Bibliothek ist eine Sammlung von Funktionen, die reguläre Ausdrücke in Perl 5 Syntax ermöglichen, Seite 54
- PNG Das *Portable Network Graphics*-Format ist ein freier Standard zur verlustfreien Kompression von Grafikdateien, Seite 45
- RFC Die *Request For Comment* spezifizieren die Datenübertragung im TCP/IP-Netzwerk, Seite 14
- Rrdtool *Rrdtool* ist ein Programm zur Auswertung und graphischen Auswertung zeitabhängiger Daten, Seite 54
- TCP Das *Transmission Control Protocol* dient der fehlerfreien Datenübertragung im Netzwerk, Seite 6
- TCP Flags Optionen im TCP Header, Seite 20

Tcpdump Programmierschnittstelle, die es ermöglicht, Daten direkt an der Netzwerkkarte auszulesen, Seite 54

TTL Die *Time To Live* beschreibt, wie viele Computer ein Datenpaket durchlaufen darf, bevor es als unzustellbar verworfen wird, Seite 17

B. Tabellen- und Abbildungsverzeichnis

Tabellenverzeichnis

3.1. Relevante Partien des TCP und IP-Headers	41
---------------------------------------------------------	----

Abbildungsverzeichnis

1.1. Vergleich Layer ISO/OSI-TCP/IP	13
1.2. Encapsulation	14
1.3. IP-Header	17
1.4. TCP-Header	19
1.5. TCP/IP Handshake	22
2.1. IP-Header, relevante Partien markiert	26
2.2. TCP-Header, relevante Partien markiert	30
2.3. TCP Optionen	34
2.4. TCP Optionen nach RFC1644	35
3.1. Echtzeitauswertung angefragter Ports	49
3.2. Gesamtanalyse angefragter Ports	50
3.3. Auswertung anfragender Betriebssysteme	51