Fachhochschule Wedel

Master-Thesis

Aufbau eines virtuellen privaten Netzes mit Peer-to-Peer-Technologie

Wolfgang Ginolas

14. August 2009

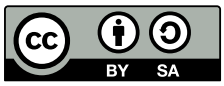
Eingereicht von:

Wolfgang Ginolas
Dipl. Ing. (FH)

E-Mail: wolfgang.ginolas@gwif.eu

Betreuer:

Prof. Dr. Ulrich Hoffmann
Fachhochschule Wedel
Feldstraße 143
22880 Wedel, Germany
Tel.: +49 4103 8048-41
E-Mail: uh@fh-wedel.de



Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Infrastruktur des Internets ermöglicht es, einfach und kostengünstig Daten zwischen verschiedenen Orten zu übertragen. Das Internet ist allerdings nicht sicher. Angreifer haben vielfältige Möglichkeiten Datenübertragungen anzuhören oder zu verändern. Wenn sich die Teilnehmer einer Datenverbindung kennen und vertrauen, sind virtuelle private Netzwerke ein mögliches Werkzeug, um sichere Verbindungen herzustellen.

Eine weitverbreitete VPN-Software ist z. B. *OpenVPN*[18]. Sie hat wie viele andere Lösungen die folgenden Eigenschaften:

- OpenVPN hat eine Client/Server-Topologie.
- Der unbefugte Zugriff auf das Netzwerk und die übertragenen Daten wird verhindert.
- Es gibt flexible aber dadurch komplizierte Konfigurationsmöglichkeiten.
- OpenVPN ist quelloffen.

OpenVPN eignet sich so z. B. für den Einsatz in Firmen, in denen die Mitarbeiter die Möglichkeit haben sollen, von außerhalb auf das Firmennetz zuzugreifen. Hier sind die Ressourcen vorhanden, einen Server durchgehend bereitzustellen und zu konfigurieren. Außerdem sind hier auch die flexiblen Einstellungsmöglichkeiten hilfreich. Zusätzlich erhöht die Verfügbarkeit der Quellen das Vertrauen in die Sicherheit von OpenVPN, da so mehr Personen die Software auf Fehler überprüfen können.

Auch für den Aufbau von VPNs zwischen Privatpersonen gibt es angepasste Lösungen, die aber entweder nicht leicht zu benutzen oder nicht unbedingt vertrauenswürdig sind.

Hamachi[16] z. B. ist ein kostenloses Programm, mit dem sich einfach private Netze aufbauen lassen. Dessen quellen sind allerdings nicht offen. Außerdem hält Hamachi eine Verbindung zu einem Server des Herstellers. Der Hersteller hat technisch die Möglichkeit, auf die Netze seiner Kunden zuzugreifen.

Quelloffene Software wie z. B. *n2n*[2] ist da vertrauenswürdiger. Die aufwendige Konfiguration über die Kommandozeile richtet sich aber an Benutzer mit guten Computerkenntnissen.

Diese Lücke soll mit der im Rahmen dieser Arbeit entwickelten Software *P2PVPN* geschlossen werden. Ziel ist es eine VPN-Software zu erstellen, die sicher und einfach einzurichten ist, ohne dass Dritten vertraut werden muss. Daraus leiten sich ab, dass diese Software eine dezentrale Topologie (*Peer-to-Peer*) benötigt, da nicht jeder die Möglichkeit hat einen geeigneten Server einzurichten. Ein von Dritten bereitgestellter Server kann nicht genutzt werden, da dies ein Sicherheitsrisiko darstellen kann. Allerdings kann ein dezentrales Netz nur schwer ohne die Hilfe einer zentralen Stelle aufgebaut werden. Ziel muss es aber sein, dass diese Stelle möglichst wenig Informationen erhält.

Zusammengefasst soll P2PVPN die folgenden Eigenschaften haben:

- P2PVPN hat eine dezentrale Topologie.
- Es werden möglichst wenig Informationen an Dritte weitergegeben.
- Der unbefugte Zugriff auf das Netzwerk und die übertragenen Daten wird verhindert.
- P2PVPN lässt sich einfach konfigurieren.
- P2PVPN ist quelloffen.

Zum Zeitpunkt der Veröffentlichung dieser Arbeit wurde P2PVPN über 2500mal heruntergeladen. Mehrere Personen benutzten P2PVPN durchgehend und unterstützen das Projekt durch ausführliche Tests und gute Fehlerberichte. Dies zeigt, dass ein Bedarf an sicherer und einfach zu bedienender VPN Software besteht. Trotz des großen Angebots an VPN-Software hat P2PVPN das Potenzial eine Nische in diesem Bereich zu füllen.

2 Grundlagen

2.1 Virtuelle private Netze

Computernetze nutzen in der Regel ein physikalisches Medium, wie z. B. Kupferkabel oder den Äther, um Daten zu übertragen. Eine Ausnahme bilden hier virtuelle Netze. Diese nutzen ein anderes, schon bestehendes Netzwerk, um Daten zu übertragen.

VPNs bauen üblicherweise ein TCP/IP Netzwerk auf und nutzen dabei das Internet als Netzzugangsschicht. Dabei erzeugt die VPN-Software virtuelle Netzwerkadapter in den Betriebssystemen der Computer, die Teil des virtuellen Netzes sind. Pakete, die diesen Adapter durchlaufen, werden allerdings nicht an ein physikalisches Medium, sondern an die VPN-Software übergeben. Diese wiederum nutzt den realen Netzwerkadapter des Computers, um das Paket über das Internet zu verschicken.

Wie das Wort *privat* in „virtuelles privates Netz“ schon andeutet, ist es die Aufgabe der VPN-Software den Zugriff auf das virtuelle Netz zu beschränken. Unbefugte dürfen weder an dem Netz teilnehmen, noch darf der Netzwerkverkehr mitgelesen werden können.

Eine kurze Beschreibung gibt es auch in [23, S. 779f].

2.2 Dezentrale Netze

Die überwiegende Mehrheit der Netzwerkprotokolle basiert auf einer Client-Server Architektur. Server bieten Dienste an, die von Clients in Anspruch genommen werden. Hieraus folgt eine klare Hierarchie und Aufgabenteilung, die das Entwickeln und Implementieren solcher Protokolle vereinfacht.

Dezentrale[1] Netze hingegen bestehen aus möglichst gleichberechtigten Knoten, die

[1] Auch *Peer-To-Peer* oder *P2P*

zusammen eine Aufgabe erfüllen. Die wohl bekannteste Anwendung für dezentrale Netze ist die Verbreitung großer Dateien, was auch *Filesharing* genannt wird. Soll eine Datei mit einem Client-Server-Protokoll an viele Clients verteilt werden, so ist die Aufgabenteilung klar: Der Server sendet und die Clients empfangen. In einem dezentralen Netz hingegen hat jeder Knoten beide Aufgaben. Daraus ergeben sich zwei Vorteile. Die Last des Sendens verteilt sich auf alle Knoten und es wird kein Server benötigt. Dadurch entfallen der Aufwand und die Kosten für den Betrieb eines Serves. Außerdem ist ein dezentrales Netz ausfallsicherer, da es keinen zentralen Punkt gibt, auf den alle angewiesen sind.

Allerdings gibt es bei der Implementierung dezentraler Netze mehrere Probleme, auf die im Folgenden eingegangen wird.

2.2.1 Bootstrapping

Damit die Knoten eines dezentralen Netzes zusammenarbeiten können, müssen sie Kommunikationsverbindungen zueinander aufbauen. Dabei ergibt sich das Problem, dass die Knoten sich nicht kennen und so auch nicht wissen, wie sie sich erreichen. Allerdings können Knoten, die bereits verbunden sind, die Adressen anderer Knoten austauschen, um weitere Verbindungen aufzubauen.

Ein neuer Knoten, der einem bestehenden Netzwerk beitreten möchte, muss mindestens einen anderen Knoten kennen, um eine Verbindung herzustellen. Verfahren, die dieses Problem lösen, werden unter dem Begriff *Bootstrapping* zusammengefasst. Nach [9, S. 155f] gibt es hier vier Möglichkeiten:

1. Ein Bootstrap-Server kennt und verteilt die Adressen aller Knoten.
2. Knoten finden andere Knoten über Broadcasts oder Multicasts.
3. Knoten finden andere Knoten über ein übergelagertes dezentrales Netz.
4. Knoten verwenden eine zwischengespeicherte Adressliste der letzten Sitzung.

Die Möglichkeiten 2. und 3. kommen allerdings für viele Anwendungen nicht infrage. Knoten, die im Internet verteilt sind, erreichen sich nicht durch Broadcasts oder Multicasts und ein passendes dezentrales Netz ist selten verfügbar. Für kleine Netze ist die zwischengespeicherte Adressliste unzuverlässig. Bei wenigen Knoten ist die Wahrscheinlichkeit hoch, dass alle ihre Adresse nach kurzer Zeit geändert haben. So bleibt oft nur die serverbasierte Lösung.

2.2.2 Network Address Translation

NAT ist eine Methode, mit der sich ganze Netze hinter einer öffentlichen IP-Adresse verbergen können. Diese Technik kommt bei praktisch allen privaten Nutzern des Internets zur Anwendung, denn diese nutzen üblicherweise einen Router, der NAT einsetzt. Dadurch können diese Nutzer mit mehreren Rechnern auf das Internet zugreifen, auch wenn sie von ihrem Provider nur eine IP-Adresse zugewiesen bekommen. Außerdem erhöht NAT die Sicherheit, da technisch bedingt eingehende Verbindungen Rechner hinter dem Router ohne weiteres nicht erreichen können. Gerade dies ist allerdings für dezentrale Netze ein großes Problem. Es hat zur Folge, dass die Knoten hinter einem NAT-Router nur schwer oder gar nicht Verbindungen zueinander aufbauen können.

Glücklicherweise ist es möglich, NAT-Router so zu konfigurieren, dass bestimmte eingehende Verbindungen an bestimmte Rechner in dem privaten Netz weitergeleitet werden. Allerdings ist man dabei auf die jeweiligen Anwender angewiesen, die diese Einstellungen an ihrem Router vornehmen müssen. Besser sind hier Verfahren, die NAT-Router ohne manuelle Unterstützung überwinden können. Dabei bieten sich mehrere Möglichkeiten, die im Folgenden beschrieben werden.

Mehr zum Thema NAT ist unter anderem in [23, S. 444ff] nachzulesen.

Universal Plug and Play

Einige Router können über UPnP konfiguriert werden. Unter anderem können Anwendungen, die im lokalen Netz laufen, den Router bitten, bestimmte Verbindungen von außen an sie weiterzuleiten. Allerdings ist UPnP in den meisten Routern deaktiviert, da dies ein Sicherheitsrisiko darstellen kann.

NAT-Traversal

NAT-Router merken sich die Absender ausgehender Pakete, um Antworten auf diese Pakete an den richtigen lokalen Rechner weiterleiten zu können. Dies kann man sich zunutze machen, um eine direkte Verbindung zwischen zwei Rechnern herzustellen, auch wenn sich beide hinter einem NAT-Router befinden. Dies funktioniert am zuverlässigsten, wenn UDP als Protokoll verwendet wird.

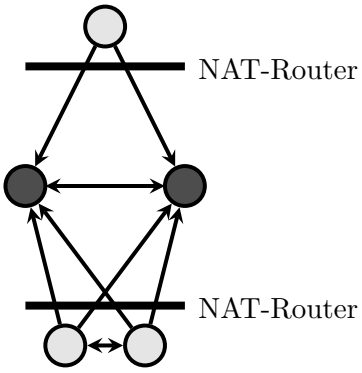


Abbildung 2.1: Beispiel für ein dezentrales Netz

Senden sich die beiden Rechner gleichzeitig Pakete, so gehen die NAT-Router davon aus, dass die eingehenden Pakete Antworten auf die eben versendeten sind, und leiten diese entsprechend weiter. Die Verbindung ist hergestellt. Dies ist allerdings nur möglich, wenn ein Dritter den Verbindungsaufbau koordiniert. Außerdem müssen die Knoten ihre externe IP-Adresse und ihren externen Port ermitteln, wozu auch Hilfe von außen nötig ist.

Dieses Verfahren findet z. B. bei der Internet-Telefonie Anwendung. Hier wird auch das Protokoll STUN[2][3] verwendet mit dem verschiedene NAT-Typen und externe IP-Adressen und Ports ermittelt werden können.

Daten weiterleiten lassen

Die zuverlässigste Methode um Daten durch NAT-Router hindurch entgegenzunehmen ist es, diese durch einen Dritten weiterleiten zu lassen. Hier bietet sich ein Knoten im dezentralen Netz an, der von allen erreicht werden kann. Dieser leitet dann Nachrichten zwischen Knoten weiter, die sich nicht direkt erreichen können.

Ein Beispiel eines Netzes, das sich auf diese Weise bildet, ist in der Abbildung 2.1 zu sehen. Es gibt in einem solchen Netz zwei Arten von Knoten, die sich darin unterscheiden, ob sie Verbindungen entgegen nehmen können oder nicht. Eine Ausnahme bilden Knoten, die gemeinsam in einem privaten Netz liegen. Diese können sich gegenseitig erreichen, auch wenn sie nicht von außen erreicht werden können. Unter der Annahme,

[2] Simple Traversal of UDP Through NATs

dass alle Verbindungen aufgebaut werden, die möglich sind, liegt zwischen zwei beliebigen Knoten immer höchstens ein Knoten. Diese Eigenschaft ist wichtig, wenn ein Verfahren zum Weiterleiten von Nachrichten für ein solches Netz entwickelt wird.

2.2.3 Routingverfahren

Routingverfahren für Peer-To-Peer-Netze beschäftigen sich in der Regel mit dem Problem, Informationen oder Knoten in einem großen weltumspannenden Netz zu finden. Da P2PVPN kleine Netze bildet, sind diese Verfahren ungeeignet.

Geeigneter sind hier Verfahren für Ad-Hoc-Funknetzwerke. Diese beschäftigen sich damit, Pakete über bestehende Verbindungen zum Ziel weiterzuleiten. Dabei werden die zum Routen notwendigen Informationen über die Netzwerktopologie automatisch gewonnen und verteilt. Mehr über diese Verfahren kann in [20, S. 77] nachgelesen werden.

Generell wird hier zwischen drei Arten von Routingverfahren unterschieden:

Proaktive Routingverfahren halten die zum Routen nötigen Informationen ständig aktuell, sodass zu jeder Zeit jeder Knoten erreicht werden kann.

Reaktive Routingverfahren ermitteln die zum Routen nötigen Informationen erst dann, wenn sie gebraucht werden.

Hybride Routineverfahren sind eine Mischung aus Proaktiven und Reaktiven Verfahren.

Proaktive Verfahren erzeugen einen größeren Overhead. Vor allem dann, wenn sich die Netzwerktopologie oft ändert. Reaktive Verfahren hingegen verschicken Pakete mit einer großen Verzögerung, wenn die Route zum Ziel erst ermittelt werden muss.

Reaktive Verfahren eignen sich also dann, wenn sich die Topologie oft ändert, und nicht jeder Knoten die Routen zu jedem anderen Knoten kennen muss.

Wenn die Vorteile beider Verfahren benötigt werden, wird ein Hybrides Routingverfahren benutzt.

2.2.4 Token-Bucket-Algorithmus

Ein TCP/IP-Netzwerk bietet nur sehr eingeschränkte Möglichkeiten, den Netzwerkverkehr zu priorisieren. Um die Internetverbindung nicht zu überlasten, bieten einige Programme die Möglichkeit an, ihre Bandbreite zu begrenzen. Vor allem bei Programmen, die ein dezentrales Netz aufbauen und ggf. Daten für andere weiterleiten, ist dies üblich. Der Token-Bucket-Algorithmus ist eine Methode zur Bandbreitenbegrenzung.

Um einen Datenfluss zu begrenzen, stellt man sich ein Gefäß („Bucket“) vor. Dieses Gefäß hat einen minimalen (0) und einen maximalen Füllstand (b). Solange es nicht voll ist, wird es mit einem konstanten Tokenstrom gefüllt. Für jedes Byte, das übertragen wird, wird ein Token aus dem Gefäß entnommen. Ist das Gefäß leer, kann nichts übertragen werden. Dies ist dann erst wieder möglich, wenn sich Tokens im Gefäß befinden.

Der Algorithmus hat also als Parameter die Geschwindigkeit des Tokenstroms und b. Die Geschwindigkeit des Tokenstroms entspricht dabei der maximalen mittleren Bandbreite. Werden viele Daten an einem Stück übertragen, kann die maximale Bandbreite kurzzeitig überschritten werden. Dabei werden aber nicht mehr als b Bytes mit einer überhöhten Bandbreite übertragen.

Eine genauere Beschreibung von dem Token-Bucket-Algorithmus ist in [23, S. 402ff] zu finden.

2.3 Kryptografie

2.3.1 Protokolle

> Ein Protokoll dient der Durchführung einer bestimmten Aufgabe und besteht aus einer Folge von Aktionen, an denen zwei oder mehr Parteien beteiligt sind. [22, S. 25]

Kryptografische Protokolle werden z. B. verwendet, um Vertraulichkeit, Integrität oder Authentizität zu schaffen. Dabei werden kryptografische Algorithmen verwendet, die unten beschrieben werden. Um Protokolle formal zu beschreiben, wird üblicherweise die folgende Symbolik verwendet:

$A \rightarrow B : M$	A Sendet die Nachricht M an B.
$E_K(M)$	M wird mit dem Schlüssel K verschlüsselt.
$D_K(C)$	C wird mit dem Schlüssel K entschlüsselt.
x, y	Konkatenation von x und y.
$H(x)$	Hashwert von x.

2.3.2 TLS

TLS[3] ist ein Protokoll, das auf TCP aufbaut und es Anwendungen ermöglicht, den Kommunikationspartner zu authentifizieren und privat mit ihm zu kommunizieren. TLS besteht aus zwei Schichten. Das „TLS Record Protocol“ ist die untere Schicht und sichert die Verbindung mit einer symmetrischen Chiffre gegen das Abhören. Das „TLS Handshake Protocol“ liegt darüber und übernimmt den Schlüsselaustausch und die Authentifizierung.

Die Authentifizierung ist für diese Arbeit interessant und wird hier kurz zusammengefasst. Eine genauere Beschreibung gibt es in [14, S. 722ff] und [5].

Um sich bei der Gegenseite zu authentisieren, wird ein Zertifikat benötigt, das die eigene Identität bestätigt. Ein solches Zertifikat wird von einer Zertifizierungsstelle erzeugt, dem die Gegenseite vertrauen muss. Eine gegenseitige Authentifizierung ist möglich, wobei in der Praxis oft nur der Client die Identität des Servers überprüft.

Vereinfacht werden bei einem Verbindungsaufbau die folgenden Schritte durchgeführt:

1. Client und Server einigen sich über die zu verwendenden Algorithmen.
2. Die Zertifikate werden ausgetauscht, soweit sie vorhanden sind.
3. Ein Schlüssel für die symmetrische Chiffre wird vereinbart.
4. Es wird überprüft, ob die Verbindung erfolgreich hergestellt wurde.

2.3.3 Advanced Encryption Standard

AES ist eine Blockchiffre, die als Nachfolger für DES[4] entwickelt wurde. Der Algorithmus wurde von Joan Daemen und Vincent Rijmen entwickelt und im Oktober 2000 vom National Institute of Standards and Technology als Standard bekannt gegeben.

[3] Transport Layer Security
[4] Data Encryption Standard

AES verschlüsselt Blöcke mit einer Länge von 128 Bit. Mögliche Schlüssellängen sind 128, 196 oder 256 Bit. [17, S. 123f]

2.3.4 RSA

RSA[22, S. 531ff] ist ein asymmetrisches Kryptosystem, das von Ron Rivest, Adi Shamir und Leonard Adleman entwickelt wurde. Es eignet sich sowohl zur Verschlüsselung als auch zum Signieren.

Implementierungen von RSA sind relativ langsam, sodass es sich nicht für große Datenmengen eignet. Deswegen wird RSA meist mit einer symmetrischen Chiffre bzw. einem Hashverfahren kombiniert.[22, S. 38ff]

2.3.5 Cipher Block Chaining Mode

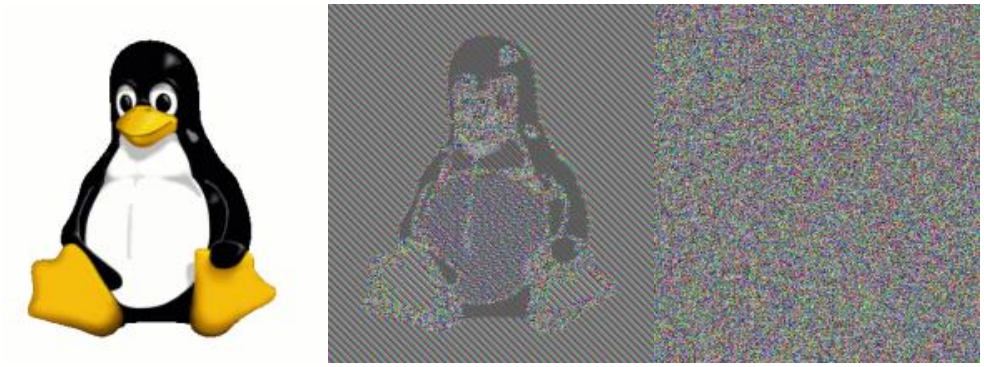
Um größere Datenmengen mit einer Blockchiffre zu verschlüsseln, muss man diese in einzelne Blöcke aufteilen. Verschlüsselt man diese Blöcke unabhängig voneinander nennt man dieses Verfahren ECB[5][22, S. 223ff]. Abbildung 2.2b zeigt beispielhaft das Ergebnis einer Verschlüsselung der Abbildung 2.2a mit ECB. Es ist offensichtlich, dass der Geheimtext Schlüsse auf den Klartext zulässt. Formal lässt sich ECB folgendermaßen beschreiben:

Verschlüsselung: $C_i = E_K(P_i)$
Entschlüsselung: $P_i = D_K(C_i)$

Jeder Block Klartextblock P_i ist genau einem Chiffretextblock C_i zugeordnet.

Ein sichereres Verfahren ist CBC[6][22, S. 227ff]. Hier geht der Geheimtext eines Blocks in die Verschlüsselung aller nachfolgenden Blöcke mit ein. Hierfür wird vor der Verschlüsselung eines Blocks der vorherige Geheimtextblock mit dem aktuellen Klartextblock verknüpft. Dadurch sind im Geheimtext Muster des Klartextes nicht mehr zu erkennen, wie Abbildung 2.2c zeigt. Allerdings müssen die Blöcke in der gleichen Reihenfolge entschlüsselt werden, wie sie verschlüsselt wurden. Außerdem dürfen bei der Entschlüsselung keine Blöcke fehlen. Formal ist CBC wie folgt definiert:

[5] Electronic Codebook Mode
[6] Cipher Block Chaining Mode



(a) Klartext (b) ECB (c) CBC

Abbildung 2.2: Notwendigkeit vom Cipher Block Chaining Mode [25]

Verschlüsselung: $C_i = E_K(C_{i-1} \oplus P_i)$
Entschlüsselung: $P_i = C_{i-1} \oplus D_K(C_i)$

Da es für den ersten Block keinen Vorgänger gibt, um ihn zu ver-/entschlüsseln, wird hier ein sogenannter *Initialisierungsvektor* verwendet. Dieser muss beim Ver- und Entschlüsseln bekannt sein, und sollte zufällig gewählt werden. Ein zufälliger Initialisierungsvektor garantiert unter anderem, dass der gleiche Klartext bei jeder Verschlüsselung zu einem anderen Geheimtext führt. Dabei ist es nicht wichtig, dass der Initialisierungsvektor geheim ist. Ein öffentlicher Initialisierungsvektor schränkt die Sicherheit von CBC nicht ein [22, S. 229]. Das Verfahren der Verschlüsselung wird in der Abbildung 2.3 gezeigt.

2.3.6 Padding

Nutzt man eine Blockchiffre, hat man normalerweise den Fall, dass die Größe der zu verschlüsselnden Daten nicht durch die Blockgröße teilbar ist. Der letzte Block kann also nicht komplett mit Daten aufgefüllt werden. Eine Blockchiffre kann allerdings nur komplette Blöcke verschlüsseln, sodass der letzte Block aufgefüllt werden muss. Dieses Auffüllen wird *Padding* genannt. Dabei muss sichergestellt werden, dass der Empfänger die Datenbytes von den Füllbytes unterscheiden kann.

ISO 10126 beschreibt ein Padding, in dem das letzte Byte des letzten Blocks die Anzahl der aufgefüllten Bytes angibt. Die noch nicht gesetzten Bytes des letzen Blocks

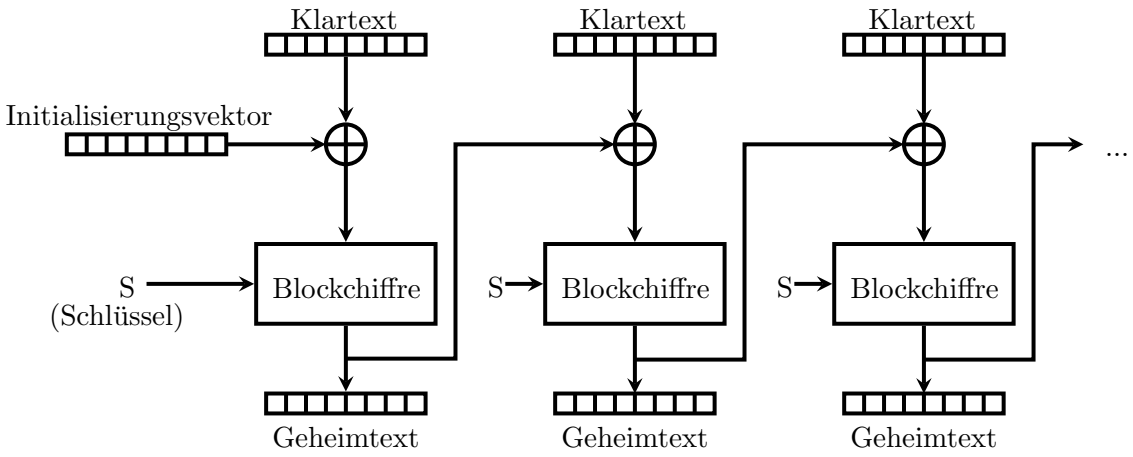


Abbildung 2.3: Cipher Block Chaining Mode

werden mit Zufallszahlen gefüllt. Es kann dabei nötig sein, einen zusätzlichen Block zu übertragen, wenn die Datenbytes den letzten Block komplett auffüllen.

3 Problemanalyse

Die Anforderungen, die sich an eine sichere und einfach bedienbare VPN Software stellen, lassen sich in mehrere Bereiche aufteilen. Diese werden im Folgenden beschrieben.

3.1 VPN

Eine VPN-Software kann Pakete auf der *Internetschicht* oder auf der *Netzzugangsschicht* abgreifen. Auch wenn das Abgreifen auf der Netzzugangsschicht einen geringfügig höheren Overhead hat, hat dies den Vorteil, dass sich die VPN-Software wie ein virtueller Ethernet-Adapter verhält. Das VPN funktioniert dadurch unabhängig von dem verwendeten Protokoll, wie z. B. IPv4, IPv6 oder IPX. Außerdem kennen die meisten Benutzer Ethernet-Adapter, was die VPN-Software verständlicher macht.

Eine VPN-Software, die die Netzzugangsschicht nutzt, verwendet die MAC-Adressen in den Paketen um diese an die richtigen Rechner weiterzuleiten. IP-Adressen sind uninteressant und werden ignoriert. Trotzdem sollte P2PVPN den Benutzer bei der Auswahl und dem Setzen der IP-Adresse unterstützen, um die Benutzerfreundlichkeit zu erhöhen.

3.2 Das dezentrale Netzwerk

Wie bereits in der Einleitung beschrieben, ist eine dezentrale Netzstruktur nötig, damit die Software einfach zu bedienen und sicher ist. Es kann von einem Nutzer nicht erwartet werden, dass er die Möglichkeiten und das Wissen hat, einen zentralen Server einzurichten. Das Nutzen eines, von Dritten eingerichteten Servers, schränkt die Sicherheit ein.

Daraus ergeben sich die, für dezentrale Netzwerke üblichen, Probleme. P2PVPN muss andere Knoten im Internet finden und Verbindungen zu ihnen aufbauen können. Bei den Netzwerken, die P2PVPN aufbaut, handelt es sich um kleine Netze. Wie schon in 2.2.1 beschrieben, wird ein zentraler Dienst benötigt, um andere Knoten zu finden. Der Nutzer sollte hier, abhängig von den eigenen Möglichkeiten, die Wahl haben, einen eigenen oder einen fremden zentralen Dienst zu nutzen.

Um Verbindungen trotz der heute üblichen NAT-Router aufzubauen, gibt es kein ideales Verfahren (siehe 2.2.2). Es ist deswegen sinnvoll jedes der folgenden Verfahren zu implementieren.

- Der Router wird automatisch konfiguriert, falls er dieses zulässt.
- Router werden mittels NAT-Traversal durchdrungen.
- Daten werden von anderen Knoten weitergeleitet.

Auf diese Weise kann die Wahrscheinlichkeit, dass Knoten eine Verbindung zueinander aufbauen können, auf ein Maximum erhöht werden.

Um Daten weiterleiten zu können, wird ein Routingverfahren benötigt. Dabei ist zu beachten, dass auf der Netzzugangsschicht in der Regel regelmäßig Broadcasts verschickt werden. P2PVPN muss diese Pakete an alle anderen Knoten senden. Normalerweise muss also jeder Knoten die Routen zu allen anderen Knoten kennen. Aus diesem Grund eignet sich ein proaktives Routingverfahren.

3.3 Sicherheit

Ein Angreifer, der physikalisch mit anderen Systemen verbunden ist, hat diverse Angriffsmöglichkeiten. Selbst Man-in-the-Middle-Angriffe sind z. B. möglich, wenn sich der Angreifer in dem selben Subnetz befindet [24]. Dies gilt selbstverständlich auch für virtuelle Netze. D. h. ein effektiver Schutz gegen einen Angreifer, der sich bereits im (virtuellen) lokalen Netz befindet, ist kaum möglich. Ziel muss es also sein, den Zugang zu dem eigenen Netz nur vertrauenswürdigen Personen zu gestatten. Außerdem sollte es möglich sein, Personen den Zugang wieder zu entziehen.

Um den Zugriff Unbefugter auf das Netzwerk zu verhindern reicht es nicht aus, den Zugang zu dem Netz einzuschränken. Es muss auch gewährleistet sein, dass der Netzwerkverkehr zwischen den Knoten nicht abgehört oder verändert werden kann. All dies

muss dezentral erfolgen, d. h. Knoten müssen sich gegenseitig überprüfen können, ohne dass eine zentrale Instanz verfügbar ist.

3.4 Weitere Anforderungen

Um von vielen Teilnehmern nutzbar zu sein, muss P2PVPN plattformunabhängig sein. Hier bietet sich die Programmiersprache Java an. Im Gegensatz zu anderen Programmiersprachen ist auch die Programmierung einer grafischen Oberfläche bei Java plattformunabhängig. Außerdem sind Java Programme üblicherweise sicherer als z. B. C Programme, da verbreitete Fehler wie Pufferüberläufe in Java nicht möglich sind.

Allerdings arbeitet eine VPN-Software sehr nah am Betriebssystem. Java kann auf benötigte Schnittstellen nicht direkt zugreifen. Ein plattformabhängiger Teil, der in C programmiert ist, ist also zwingend notwendig. Außerdem ist Java langsamer als z. B. C. Inwieweit das ein Problem ist, wird im Kapitel 6.2.3 beschrieben.

4 Entwurf

4.1 VPN

Unix-artige Betriebssysteme bieten für den Zugriff auf virtuelle Netzwerkadapter die sogenannten „TUN“- und „TAP“-Schnittstellen an[13]. TUN arbeitet dabei auf der Internetschicht, TAP auf der Netzzugangsschicht. Je nach verwendetem Unix wird der virtuelle Netzwerkadapter durch eine Datei wie z. B. `/dev/tun0` oder `/dev/net/tun` repräsentiert. Durch das Schreiben und Lesen dieser Datei werden Netzwerkpakete an den TCP/IP-Stack des Betriebssystems gesendet, oder von ihm empfangen.

Windows enthält keine TUN/TAP-Schnittstelle. Allerdings wurde im Rahmen des OpenVPN-Projektes ein Treiber für einen virtuellen Netzwerkadapter entwickelt, der eine Schnittstelle hat, die TAP ähnelt. Ein Treiber, der wie TUN auf der Internetschicht arbeitet, existiert nicht.

TAP kann also gut für eine plattformübergreifende VPN-Software genutzt werden. Allerdings unterscheiden sich die Schnittstellen leicht, sodass ein kleiner Teil der Software für jedes Betriebssystem getrennt implementiert werden muss.

4.2 Das dezentrale Netzwerk

4.2.1 Schichten

Wie in der Abbildung 4.1 zu sehen ist, gliedert sich P2PVPN zwischen den Netzwerkschichten des physikalischen und des virtuellen Netzwerks ein. P2PVPN ist dabei selbst in verschiedene Schichten aufgeteilt, die ein dezentrales Netzwerk aufbauen. Diese Schichten erfüllen die folgenden Aufgaben:

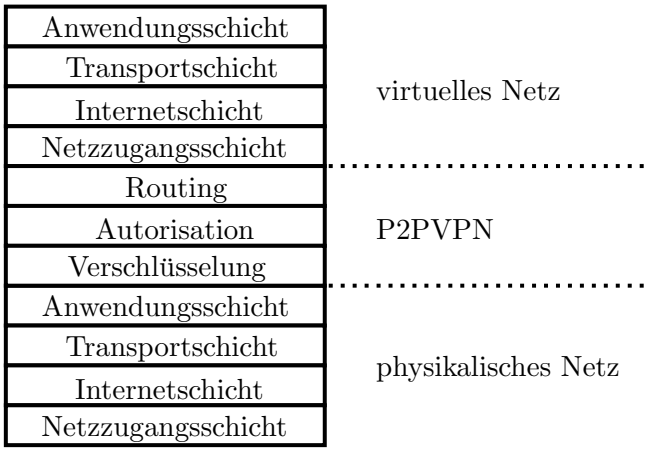


Abbildung 4.1: Die am VPN beteiligten Schichten

Verschlüsselung Diese Schicht dient dazu, Pakete effizient über eine TCP-Verbindung zu versenden. Zusätzlich werden die Pakete hier mit einem symmetrischen Verfahren verschlüsselt.

Autorisierung Die *Autorisierungsschicht* entscheidet bei jedem Verbindungsaufbau, ob der neue Knoten an diesem Netzwerk teilnehmen darf.

Routing Die *Routingschicht* hat u. a. die Aufgabe, Pakete von dem virtuellen Netzwerkadapter an die richtigen Knoten zu senden, oder Pakete von anderen Knoten weiterzuleiten. Dadurch können Knoten sich auch dann Pakete schicken, wenn sie nicht direkt verbunden sind.

Die Abbildung 4.2 zeigt, wie die Schichten zusammenspielen, um ein Paket zu übertragen. In diesem Beispiel sendet der Knoten A ein Paket an C. Da A nicht direkt mit C verbunden ist, wird das Paket durch B weitergeleitet.

Im Folgenden wird genauer auf die Aufgaben der einzelnen Schichten eingegangen.

Verschlüsselung

Zusätzlich zu der Verschlüsselung, die im Kapitel 4.3 beschrieben ist, hat die Verschlüsselungsschicht die Aufgabe, Datenpakete über eine TCP-Verbindung zu übertragen. Das ist nicht so trivial, wie es zuerst den Anschein hat, denn es ergibt sich dabei das folgende Problem.

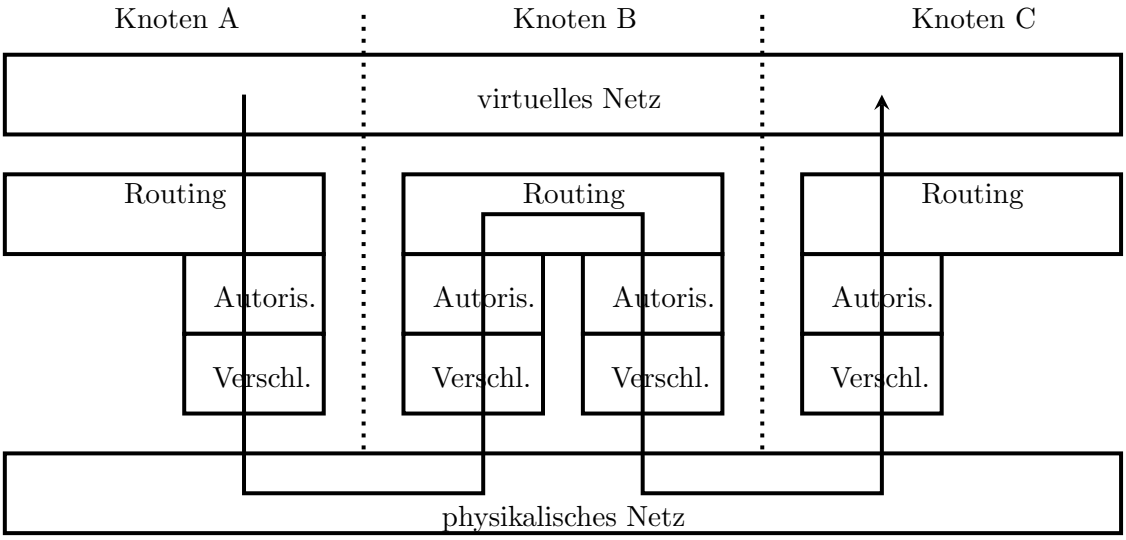


Abbildung 4.2: Ein weitergeleitetes Paket

Ein Ethernet-Paket kann maximal 1500 Bytes an Nutzdaten enthalten und hat dabei eine Gesamtlänge von 1514 Bytes. Berücksichtigt man die üblichen Größen der Header, so ergibt sich, dass ein TCP-Paket maximal 1460 Bytes an Nutzdaten enthalten kann, wenn es über Ethernet versendet wird.

Da die Nutzdaten meistens voll ausgenutzt werden, liest P2PVPN normalerweise 1514 Byte große Pakete von dem virtuellen Netzwerkadapter. Diese erweitert P2PVPN um weitere Informationen, sodass man üblicherweise auf 1522 Bytes kommt. Diese 1522 Bytes des virtuellen Paketes müssen nun mit 1460 Byte großen physikalischen TCP-Paketen über das Internet verschickt werden. Aus Performancegründen sollten die TCP-Pakete komplett aufgefüllt sein. Außerdem ist darauf zu achten, dass die virtuellen Pakete mit einer geringen Zeitverzögerung verschickt werden.

Um dies zu erreichen, wird für jede TCP-Verbindung ein Thread gestartet, der dafür zuständig ist, Pakete zu versenden. Die zu versendeten Pakete entnimmt der Thread einer Warteschlange. Solange die Warteschlange gefüllt ist, verschickt der Thread die Pakete so schnell, wie es geht. Das Betriebssystem kümmert sich dabei darum, den Datenstrom in möglichst große TCP-Pakete aufzuteilen. Sobald die Warteschlange leer ist, wird der Puffer der TCP-Verbindung mit einem `flush` geleert, damit auch das letzte Paket den Empfänger vollständig erreicht.

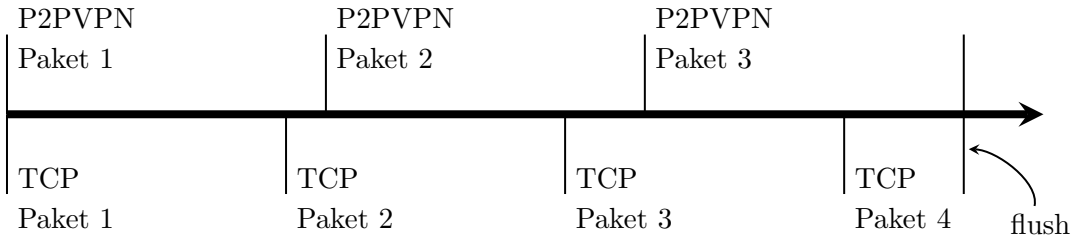


Abbildung 4.3: Das Versenden von P2PVPN-Paketen über leicht kleinere TCP-Pakete

So werden zwei Dinge erreicht. Wenn die Verbindung ausgelastet, d. h. die Warteschlange gefüllt ist, werden die TCP-Pakete komplett ausgefüllt. Dadurch wird der Overhead möglichst gering gehalten. Wenn einzelne Pakete versendet werden, ist die Warteschlange sofort wieder leer, sodass diese Pakete ohne Verzögerung versendet werden. In Abbildung 4.3 ist ein Beispiel zu sehen, in dem drei P2PVPN-Pakete übertragen werden. Dazu werden vier TCP-Pakete benötigt. Die ersten drei sind voll aufgefüllt. Das Vierte wird abgeschickt, bevor es voll ist, damit auch das dritte P2PVPN-Paket ohne Verzögerung vollständig übertragen wird.

Die Warteschlange hat eine maximale Länge. Wenn die Warteschlange voll ist, werden zu sendende Pakete verworfen. Dies ist das übliche Vorgehen, das auch von Switches und Routern verwendet wird, wenn eine Leitung überlastet ist. Der virtuelle TCP/IP-Stack wird dies bemerken, und die Senderate entsprechend drosseln.

Möchte der Benutzer die Bandbreite die P2PVPN nutzt, zusätzlich drosseln, müssen zusätzliche Pakete verworfen werden. Dies wird die Senderate weiter senken. Um zu entscheiden, welche Pakete verworfen werden, wird der Token-Bucket-Algorithmus verwendet.

Es hat sich gezeigt, dass das Begrenzen der Empfangsrate nicht zuverlässig funktioniert. Aus diesem Grund kann momentan nur die Senderate begrenzt werden. Für die meisten Anwendungsfälle sollte dies allerdings ausreichen, da das Senden bei Internetverbindungen üblicherweise den Flaschenhals darstellt. In Kapitel 5.2.3 wird das Problem genauer beschrieben.

Autorisierung

Die Autorisierungsschicht überprüft, ob ein anderer Knoten an diesem P2PVPN-Netz

teilnehmen darf. Diese Überprüfung findet beim Verbindungsaufbau statt, und wird in dem Kapitel 4.3.2 beschrieben.

Jede Verbindung wird auf diese Weise überprüft. Dadurch müssen sich zwei Knoten, die nicht direkt verbunden sind, nicht autorisieren. Es genügt, dass ein anderer Knoten die Autorisierung durchgeführt hat.

Außerdem wird hier der Schlüssel für die Verschlüsselungsschicht vereinbart.

Routing

Da nicht immer jeder Knoten eine Verbindung zu jedem anderen aufbauen kann (siehe 2.2.2), wird die Routingschicht benötigt. Sie kann Pakete für Knoten weiterleiten, die nicht direkt verbunden sind. Der Routingalgorithmus basiert auf einer verteilten Datenbank (siehe 4.2.2). Die Funktionsweise der Routingschicht wird in 4.2.3 genau beschrieben.

4.2.2 Verteilte Datenbank

Der Routingalgorithmus und andere Funktionen von P2PVPN basieren auf einer verteilten Datenbank, die es jedem Knoten ermöglicht Informationen zu verbreiten. Diese Daten werden über das gesamte Netzwerk verteilt und aktuell gehalten. Dabei kann jeder Knoten nur Daten setzen oder verändern, die ihn betreffen. Lesbar sind diese Daten für alle.

Die Daten werden in assoziativen Arrays abgelegt, in denen die Schlüssel und die Werte von Typ `String` sind. So bleibt die Datenstruktur einfach und lässt sich dennoch flexibel einsetzen. Für jeden Knoten gibt es ein Array, das von ihm verändert werden kann. Eine Kopie dieses Arrays liegt in jedem anderen Knoten. So kenn jeder Knoten die gesamten Daten.

Um die Synchronisation zu vereinfachen, hat jedes Array eine Versionsnummer, die sich bei einer Änderung erhöht. Die Synchronisation selbst basiert auf Polling. Um die Kopien der Arrays aktuell zu halten, fragt jeder Knoten regelmäßig seine Nachbarn, ob sie eine aktuellere Version haben. Diese antworten dann ggf. mit einer aktuellen Kopie des betreffenden Arrays. Um Knoten bzw. Arrays zu identifizieren, wird die eindeutige `PeerID` der Autorisierungsschicht genutzt. Welcher Knoten für welches Array

gefragt wird, ermittelt der unten beschriebene Routingalgorithmus. Im Idealfall kann der Knoten direkt gefragt werden, von dem das Array ursprünglich stammt. Ist dies nicht möglich, wird ein geeigneter Nachbar gefragt.

Polling wird verwendet, da es auch dann zuverlässig arbeitet, wenn gelegentlich Pakete bei der Übertragung verloren gehen. Da immer das komplette Array übertragen wird, stellt bei den typischen Datenmengen kein Problem dar. Wenn in Zukunft größere Datenmengen gespeichert werden, sollte das Protokoll so erweitert werden, dass nur Änderungen übertragen werden.

4.2.3 Routing

Adressen

Um Routen zu können, müssen Knoten eindeutig identifiziert werden. Ein Knoten im P2PVPN-Netzwerk hat mehrere Arten von Adressen:

IP-Adresse des virtuellen Netzwerkadapters Die IP-Adresse spielt für P2PVPN keine Rolle, da es auf der Netzzugangsschicht arbeitet. Allerdings unterstützt P2PVPN den Benutzer bei der Wahl und dem Setzen einer IP-Adresse.

MAC-Adresse des virtuellen Netzwerkadapters Die MAC-Adresse identifiziert Knoten auf der Netzzugangsschicht und wird zum Routen verwendet.

PeerID (ID_A der Autorisierungsschicht) Die PeerID identifiziert einen Knoten eindeutig und kann nicht verändert werden. Sie wird in der verteilten Datenbank verwendet.

Da der virtuelle Netzwerkadapter Pakete auf der Netzzugangsschicht erhält, ist es sinnvoll, dass das Routing mit MAC-Adressen arbeitet. Dadurch wird der Overhead gering gehalten, da die zum Routen nötige Ziel-MAC-Adresse bereits im Paket enthalten ist. Theoretisch wäre es auch möglich, die PeerID zum Routen zu verwenden. Diese müsste dann aber in jedem Paket gespeichert werden, was dieses unnötig vergrößert.

Routingalgorithmus

Wie schon in der Problemanalyse beschrieben, verwendet P2PVPN ein proaktives Routingverfahren. Jeder Knoten speichert eine Liste der PeerIDs seiner direkten Nachbarn

in der verteilten Datenbank. So kennt jeder Knoten den kompletten Graphen, den das Netz bildet. Damit eine Zuordnung zwischen PeerID und MAC-Adresse möglich ist, speichert jeder Knoten auch seine MAC-Adresse in der Datenbank.

Routen werden so gewählt, dass die Anzahl an Knoten, die ein Paket weiterleiten müssen, minimal ist. Es wird also eine einfache Variante des *Shortest Path Routing* [23, S. 353ff] verwendet. Da es üblicherweise Knoten geben wird, die mit allen anderen Knoten verbunden sind (siehe 2.2.2), ist eine Route normalerweise nicht länger als zwei Hops.

Sind zwei Knoten nicht direkt verbunden, gibt es oft mehrere mögliche Routen mit zwei Hops. Um aus diesen Routen eine auszuwählen, ermittelt jeder Knoten regelmäßig die Latenz zu allen seinen Nachbarn. Dabei wird ein verloren gegangenes Ping-Paket als eine sehr hohe Latenz gewertet. Die Route, die über den Nachbarn mit der geringsten Latenz läuft, wird gewählt. Dies ist ein sehr einfaches Verfahren, dass aber unter den folgenden Annahmen gute Ergebnisse liefert:

1. Routen sind nicht länger als zwei Hops.
2. Der Verlust eines Paketes liegt darin begründet, dass ein Knoten auf der Route dieses Paket verworfen hat, da dessen Sendebandbreite ausgelastet ist.

Die erste Annahme trifft wie oben beschrieben in der Regel zu. Die zweite Annahme ergibt sich daraus, dass die meisten Knoten über einen normalen ADSL-Anschluss angebunden sind. Bei solchen Anschlüssen ist die Sendebandbreite deutlich geringer als die Empfangsbandbreite. Dadurch wird die Menge der Pakete, die ein Knoten weiterleiten kann, durch die Sendebandbreite begrenzt. Ist die Sendebandbreite eines Knotens überlastet, so werden seine Nachbarknoten dies erkennen, da Ping-Pakete verloren gehen. Dadurch wird dieser Knoten automatisch in Routen gemieden.

Dieses Routingverfahren hat also die folgenden Eigenschaften:

- ✓ Minimiert die Hops
- ✓ Minimiert die Latenz
- ✓ Ein Knoten mit einer langsamen Anbindung wird erkannt und in Routen gemieden.
- ✓ Fällt ein Knoten aus, werden unmittelbar andere Knoten in den Routen verwendet.

✗ Ist die Bandbreite des Netzes ausgelastet, wird die Last nicht gleichmäßig auf die Knoten verteilt.

Um Knoten, die Pakete weiterleiten, gleichmäßig auszulasten, wird es vermutlich nötig sein, Informationen, wie die maximale und momentane Bandbreite, in den Routingalgorithmus mit einzubeziehen. Dies könnte ein Bestandteil einer zukünftigen Untersuchung sein.

Broadcasts und Multicast

Ist das niederwertigste Bit in einer MAC-Zieladresse gesetzt, handelt es sich um ein Broadcast- oder Multicast-Paket [4, S. 50]. Ein solches Paket muss an alle Knoten im Netzwerk gesendet werden. Broadcasts werden von ARP[1] verwendet. Deswegen ist es zwingend notwendig, dass P2PVPN Broadcasts unterstützt. Da das Internet keine Multicasts unterstützt, muss P2PVPN solche Pakete per Hand an jeden einzelnen Knoten senden. Dieser hohe Aufwand ist vertretbar, solange das Netzwerk klein ist oder wenig Daten an alle gesendet werden müssen.

Dass P2PVPN Broadcasts und Multicasts unterstützt, hat zur Folge, dass auch Protokolle wie z. B. Bonjour[7] über das virtuelle Netzwerk funktionieren. Dies ist sehr praktisch und steigert die Benutzerfreundlichkeit.

Interne Pakete

Die Hauptaufgabe der Routingschicht ist es Pakete, die von der virtuellen Netzzugangsschicht kommen, korrekt weiterzuleiten. Allerdings wird die Flexibilität des P2PVPN-Netzes deutlich erhöht, wenn auch andere Pakete übertragen werden können, die nicht aus dem virtuellen Netzwerk stammen. Aus diesem Grund wurde die Routingschicht so erweitert, dass auch andere Komponenten von P2PVPN Pakete versenden können. Momentan nutzt der Chat diese Möglichkeit. Außerdem werden auch die Ping-Pakete auf diese Weise verschickt.

In Zukunft kann dies genutzt werden, um beliebige andere Daten über das P2PVPN-Netz zu übertragen, ohne das ein funktionierendes virtuelles Netzwerk benötigt wird.

[1] Address Resolution Protocol

4.2.4 Bootstrapping

Die externen IP-Adressen der Knoten sind in der dezentralen Datenbank gespeichert. Es reicht also, dass ein neuer Knoten mit nur einem anderen Knoten im Netz verbunden wird, um die IP-Adressen der anderen Knoten zu erfahren. Um allerdings eine Verbindung zu dem ersten Knoten aufzubauen, gibt es mehrere Möglichkeiten.

Eine einfache Möglichkeit ist es, einen oder mehrere Knoten mit einer statischen IP-Adresse oder mit einem festen DNS-Namen einrichten. Alle Knoten müssen diese Liste der „bekannten Knoten“ kennen. Solange immer ein bekannter Knoten verfügbar ist, können auch andere Knoten eine Verbindung zum Netz aufbauen. Eine Erweiterung dieser Möglichkeit stellt eine persistente Liste dar, die jeder Knoten speichert. Hier werden alle IP-Adressen festgehalten, hinter denen in der letzten Zeit ein Knoten zu finden war.

Diese Methode hat den Vorteil, das keine Dienste von Dritten in Anspruch genommen werden müssen. Das Netzwerk arbeitet deswegen relativ verborgen, und kann von Außenstehenden nur recht schwer entdeckt werden. Allerdings ist der Aufwand, einen Knoten durchgehend laufen zu lassen und einen DNS-Namen einzurichten, für einen Benutzer recht hoch. Die persistente Liste von IP-Adressen alleine ist unzuverlässig, da die IP-Adresse von den meisten Internetanschlüssen spätestens alle 24 Stunden geändert wird. Diese Methode wurde deswegen vor allem für Benutzer mit den nötigen Kenntnissen und Möglichkeiten implementiert.

Speichert man die IP-Adressen der Knoten an einer zentralen Stelle im Internet, wird das Benutzen von P2PVPN deutlich einfacher. Es sind hier viele Möglichkeiten denkbar. Zwei werden im Folgenden beschrieben.

BitTorrent

BitTorrent[10] ist ein Filesharing-Protokoll, bei dem sich mehrere BitTorrent-Knoten verbinden, um gemeinsam eine bestimmte Datei herunterzuladen. Die Knoten benutzen dabei einen sogenannten Tracker, um die IP-Adressen anderer Knoten zu erfahren. Jeder Knoten sendet dabei einen Hashwert der Datei und die eigene IP-Adresse zu dem Tracker. Dieser antwortet mit einer Liste von IP-Adressen, die dem Tracker zu diesem Hashwert bereits bekannt sind.

Es gibt mehrere solcher Tracker, die öffentlich im Internet verfügbar sind. Diese eignen sich auch, um bei dem Verbindungsaufbau von P2PVPN zu helfen. Anstelle des Hashwertes einer Datei sendet P2PVPN den Hashwert des Netzwerks, an dem es teilnehmen will. Die IP-Adressen, die der Tracker dann sendet, sind Adressen anderer P2PVPN-Knoten.

Anfragen an einen Tracker laufen über HTTP. Die Parameter der Anfrage werden mit einem `GET` übertragen. Die Antwort ist im *Bencode*-Format[26] codiert, welches eigens für BitTorrent entwickelt wurde. Mit Bencode können relativ leicht strukturierte Daten beschrieben werden. Es kennt die Datentypen Zahl, Byte-Array, Liste und assoziatives Array.

OpenDHT

OpenDHT[21] ist ein Netzwerk aus etwa 200 Rechnern, die zusammen eine verteilte Hashtabelle bilden. Auch wenn es nicht möglich ist, als Knoten an diesem Netz teilzunehmen, so bietet OpenDHT trotzdem einen öffentlichen Zugriff auf die Hashtabelle. Das Ziel von OpenDHT ist es, verteilte Hashtabellen unter realen Verhältnissen zu testen und zu erforschen.

Um Schlüssel-Wert-Paare zu speichern, bietet OpenDHT eine `put`-Funktion an, die über XML-RPC aufgerufen werden kann. Mit der Funktion `get` kann eine Menge von Werten, die unter einem Schlüssel gespeichert wurden, abgefragt werden. Schlüssel haben dabei eine maximale Länge von 20 Byte und Werte dürfen nicht länger als 1024 Byte sein. Außerdem hat jedes Schlüssel-Wert-Paar eine maximale Lebensdauer, die nicht länger als eine Woche sein kann.

Prinzipiell eignet sich OpenDHT gut, um die IP-Adressen von P2PVPN-Knoten zu speichern. Allerdings hat sich herausgestellt, dass OpenDHT sehr unzuverlässig und langsam ist. Aus diesem Grund wurde dieser Ansatz nicht weiter verfolgt.

4.3 Sicherheit

Wie in der Problemanalyse beschrieben, muss P2PVPN die Möglichkeit bieten, nur bestimmten Knoten den Zugang zu dem Netzwerk zu erlauben. In dem Schichtenmodell ist bereits zu sehen, dass diese Aufgabe in zwei Teile unterteilt ist. Die *Autorisierung*

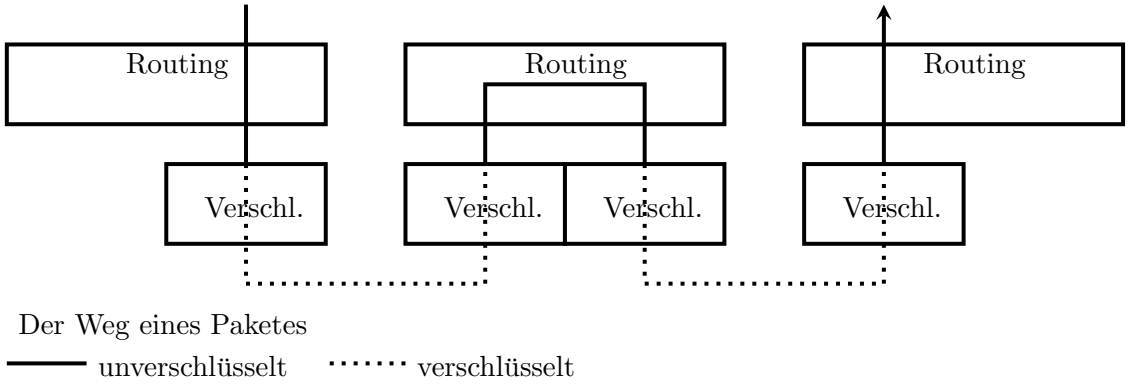


Abbildung 4.4: Knoten-zu-Knoten-Verschlüsselung

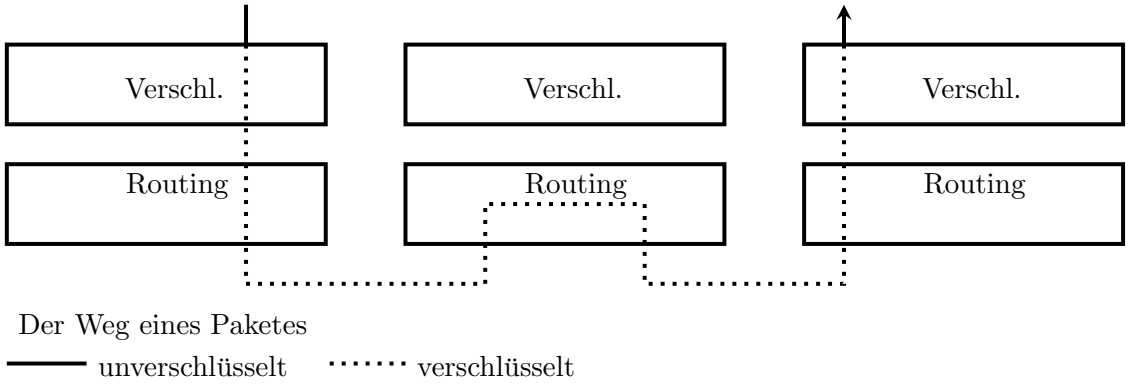


Abbildung 4.5: Ende-zu-Ende-Verschlüsselung

überprüft, ob ein Knoten am Netz teilnehmen darf. Die *Verschlüsselung* stellt sicher, dass keine Außenstehenden den Netzwerkverkehr abhören oder verändern können.

Die Verschlüsselungsschicht liegt dabei möglichst weit unten. Das hat den Vorteil, dass die Daten der darüberliegenden Schichten verschlüsselt sind und nicht abgehört werden können (siehe Abbildung 4.4). Allerdings wird das Paket auf jedem Knoten, der es weiterleitet, entschlüsselt. D. h., dass dieser Knoten das Paket mitlesen und verändern kann.

Würde die Verschlüsselungsschicht über der Routingschicht liegen (siehe Abbildung 4.5), könnte der weiterleitende Knoten das Paket nicht mitlesen. Allerdings wären dann die Routinginformationen in den Paketen (z. B. die Zieladresse) nicht verschlüsselt.

Aus diesem Grund gibt es in vielen sicheren Protokollen Verschlüsselungsschichten auf mehreren Ebenen. P2PVPN ist hier allerdings ein Sonderfall. Wie in Kapitel 3.3 beschrieben, müssen aufgrund von Sicherheitsproblemen im virtuellen Netz, alle Knoten vertrauenswürdig sein. Deswegen ist eine Verschlüsselungsschicht über der Routingschicht nicht nötig.

4.3.1 Verschlüsselung

Um die Daten zu verschlüsseln, wird AES im CBC-Modus genutzt. Beide Enden einer Verbindung benutzen den gleichen Schlüssel zum Ver- und Entschlüsseln. Jede Verbindung im P2PVPN-Netz hat einen eigenen Schlüssel, der in der Autorisierungsschicht ermittelt wird.

Der Initialisierungsvektor, der für das CBC benötigt, wird zufällig erzeugt und unverschlüsselt übertragen. Jedes Ende einer Verbindung hat einen eigenen Initialisierungsvektor.

Wie unten beschrieben ist, wird der Schlüssel während der Autorisierung einmal neu gesetzt. Die Verschlüsselungsschicht bietet deswegen die Möglichkeit, den Schlüssel einer Verbindung nachträglich zu ändern. Bei jeder Schlüsseländerung wird ein neuer Initialisierungsvektor erzeugt und übertragen. Die Autorisierungsschicht muss dabei auf die Synchronisation der Schlüsseländerung achten.

Da die verschlüsselten Pakete über TCP versendet werden, ist sichergestellt, dass keine Pakete verloren gehen und ihre Reihenfolge nicht ändern. Diese Eigenschaft ist für den CBC-Modus zwingend notwendig. Soll in Zukunft UDP für das Versenden von Paketen verwendet werden, muss die Verschlüsselungsschicht entsprechend verändert werden.

4.3.2 Autorisierung

Die Entscheidung, welche Knoten an einem Netzwerk teilnehmen dürfen, sollte an einer zentralen Stelle getroffen werden können. Wie im Kapitel 3.3 beschrieben, muss das Netzwerk aber auch dann funktionieren, wenn diese Stelle gerade nicht verfügbar ist. Ein gemeinsames Geheimnis, wie z. B. ein Passwort wäre ein erster Ansatz für eine Lösung. Bei einem Verbindungsaufbau könnten zwei Knoten überprüfen, ob sie das

gleiche Geheimnis haben und ob der jeweils andere berechtigt ist, am Netzwerk teilzunehmen. Dieses Geheimnis müsste dann an alle berechtigten Knoten weitergegeben werden. Dies könnte z. B. über E-Mail geschehen.

Es wäre dann allerdings nicht möglich, Knoten aus dem Netz auszuschließen, da man ein Geheimnis nicht wegnehmen kann. Außerdem kann das Geheimnis beliebig weitergegeben werden, sodass die zentrale Stelle ein Wachsen des Netzes nicht kontrollieren kann.

Aus diesem Grund wurde für P2PVPN ein Verfahren entwickelt, welches TLS in gewissen Punkten ähnelt. Damit Knoten beweisen können, dass sie am Netzwerk teilnehmen dürfen, bekommen sie *Einladungen*. Diese Einladungen ähneln den Zertifikaten bei TLS. Die Zertifizierungsstelle ist bei P2PVPN die Person, die das Netzwerk erstellt hat. Außerdem kann die Möglichkeit, Einladungen zu erstellen an andere weitergegeben werden. Daraus ergeben sich zwei Arten von Einladungen, die sich darin unterscheiden, ob der Eingeladene berechtigt ist, weitere Knoten einzuladen oder nicht.

TLS wird in P2PVPN nicht genutzt, da es ausschließlich mit TCP arbeitet. Dadurch wäre es ausgeschlossen, dass P2PVPN in Zukunft auch UDP zum Transport nutzen kann.

Netzwerk-Einladung

Eine *Netzwerk*-Einladung, die das Erstellen von weiteren Einladungen erlaubt, besteht aus dem folgenden 4-Tupel:

Netzwerk-Einladung: $\left(I_N, P_N, S_{S_N}(I_N, P_N), S_N\right)$

I_N	Einstellungen des Netzwerks (Name, IP-Subnetz etc.)
P_N	Öffentlicher Schlüssel des Netzwerks
S_N	Geheimer Schlüssel des Netzwerks
$S_{S_N}(I_N, P_N)$	Eine Signatur über I_N und P_N, die mit S_N erstellt wurde.

Außerdem hat jedes Netzwerk eine eindeutige Nummer, die aus der Signatur gebildet wird: $ID_N = H(S_{S_N}(I_N, P_N))$. Es ist zu beachten, dass so das Verändern von I_N verhindert wird. Würde man I_N ändern, würde sich auch die Signatur und damit ID_N verändern. ID_N muss aber für alle Knoten gleich sein, damit sie eine Verbindung aufbauen können. So wird sichergestellt, dass auch Knoten die einladen dürfen, nicht die ursprünglichen Netzwerkeinstellungen modifizieren können.

Wenn ein neues Netzwerk erstellt wird, wählt der Benutzer die Einstellungen I_N. Ein Schlüsselpaar (P_N, S_N) wird erzeugt und die Signatur wird generiert.

Zugangs-Einladung

Eine *Zugangs*-Einladung erlaubt nicht das Erstellen von weiteren Einladungen. Jeder Knoten braucht eine individuelle Zugangs-Einladung. Haben zwei Knoten die gleiche Zugangs-Einladung, so kann von den beiden immer nur einer zur Zeit am Netzwerk teilnehmen. Wenn ein Knoten eine Netzwerk-Einladung bekommen hat, muss er für sich eine eigene Zugangs-Einladung generieren.

Eine Zugangs-Einladung besteht aus dem folgenden 7-Tupel:

Zugangs-Einladung: $\Big(I_A, P_A, S_{S_N}(I_A, P_A), S_A, I_N, P_N, S_{S_N}(I_N, P_N)\Big)$

I_A	Informationen über den Knoten (z. B. Verfallsdatum der Einladung)
P_A	Öffentlicher Schlüssel des Zugangs
S_A	Geheimer Schlüssel des Zugangs
$S_{S_N}(I_A, P_A)$	Eine Signatur über I_A und P_A, die mit S_N erstellt wurde.
$I_N, P_N, S_{S_N}(I_N, P_N)$	Ein Teil der Netzwerk-Einladung

Wie auch das Netzwerk haben die Knoten eine eindeutige Nummer, die aus dem Öffentlichen schlüssel gebildet wird: $ID_A = H(P_A)$. ID_A wird in dieser Arbeit auch `PeerID` genannt. Ein Knoten kann I_A und ID_A nicht ändern, wenn er S_N nicht kennt.

Um eine Zugangs-Einladung zu erstellen, muss der Benutzer I_A wählen. Das Schlüsselpaar (P_A, S_A) und die Signatur werden automatisch erzeugt.

Verbindungsaufbau und Autorisierung

Angenommen, die Knoten X und Y erhalten die folgenden Zugangs-Einladungen:

X: $\Big(I_{AX}, P_{AX}, S_{S_N}(I_{AX}, P_{AX}), S_{AX}, I_N, P_N, S_{S_N}(I_N, P_N)\Big)$
Y: $\Big(I_{AY}, P_{AY}, S_{S_N}(I_{AY}, P_{AY}), S_{AY}, I_N, P_N, S_{S_N}(I_N, P_N)\Big)$

Das Protokoll für die Autorisierung ist wie folgt aufgebaut:

1. X und Y setzen den Schlüssel der Verschlüsselungsschicht auf ID_N.

2. $X \rightarrow Y : \Big(I_{AX}, P_{AX}, S_{S_N}(I_{AX}, P_{AX})\Big)$
3. $Y \rightarrow X : \Big(I_{AY}, P_{AY}, S_{S_N}(I_{AY}, P_{AY})\Big)$
4. X überprüft die Signatur und dessen Verfallsdatum von Y.
5. Y überprüft die Signatur und dessen Verfallsdatum von X.
6. $X \rightarrow Y : E_{S_{AX}}(R_X)$ mit (R_X = Zufallszahl)
7. $Y \rightarrow X : E_{S_{AY}}(R_Y)$ mit (R_Y = Zufallszahl)
8. X setzt den Schlüssel der Verschlüsselungsschicht auf $D_{P_{AY}}(E_{S_{AY}}(R_Y)) \oplus R_X$.
9. Y setzt den Schlüssel der Verschlüsselungsschicht auf $D_{P_{XY}}(E_{S_{AX}}(R_X)) \oplus R_Y$.

Durch Punkt 1 ist der Verbindungsaufbau von Anfang an verschlüsselt. Außerdem wird hier sichergestellt, dass sich beide Knoten sich in dem gleichen Netzwerk befinden. In 2-5 überprüfen die Knoten, ob der andere eine gültige Zugangs-Einladung hat. In 6-9 wird ein neuer Schlüssel vereinbart. Dies ist nötig, da ID_N auch Knoten bekannt ist, die eine abgelaufene Einladung haben. Diese sollen die Verbindung aber nicht abhören können.

Knoten ausschließen

Das oben erwähnte Verfallsdatum der Einladung kann benutzt werden, um Knoten nur für eine begrenzte Zeit an dem Netzwerk teilnehmen zu lassen. Ist das Verfallsdatum abgelaufen, kann ein Knoten nur dann eine Verbindung zu dem Netz aufbauen, wenn er eine neue Einladung bekommt. Vor dem Ablauf des Verfallsdatums können Knoten allerdings nicht ausgeschlossen werden und Knoten, deren Einladung kein Verfallsdatum hat, können nie ausgeschlossen werden.

Um Knoten die nur eine Zugangs-Einladung haben den Zugriff zu einem beibiegen Zeitpunkt zu verwehren, könnte eine schwarze Liste verwendet werden. Auf dieser Liste wäre von jedem ausgeschlossenen Knoten die ID_A zu finden. Diese Liste müsste mit dem privaten Netzwerkschlüssel S_N signiert sein, damit sie nicht unbefugt geändert werden kann. Da die Autorisierung dezentral stattfindet, ist ein Weg zu finden, der die Liste auf alle Knoten verteilt und synchron hält. Eine Erweiterung der dezentralen Datenbank oder ein neuer Mechanismus wird hier benötigt.

Damit die Liste mit der Zeit nicht immer länger wird, kann das Verfallsdatum in den Einladungen genutzt werden, um Einträge wieder zu entfernen. Ein Knoten mit einer abgelaufenen Einladung braucht nicht durch die Liste gesperrt zu werden.

Dieser Ansatz wurde nicht implementiert. P2PVPN kann aber entsprechend erweitert werden.

5 Implementierung

5.1 VPN

Die verfügbare Dokumentation der TAP-Schnittstelle geht kaum über [13] hinaus, sodass die nötigen Informationen für P2PVPN überwiegend durch Reverse Engineering gewonnen wurden. In der Software VTun[12] ist gut zu erkennen, wie unter Linux auf die TAP-Schnittstelle zugegriffen wird.

Für den Zugriff auf die TAP-Schnittstelle werden bei Windows und auch bei Linux Funktionen benötigt, die Java nicht bereitstellt. Aus diesem Grund kann ein reines Java-Programm nicht auf einen virtuellen Netzwerkadapter zugreifen. Es ist also notwendig, diesen Teil der VPN-Software in C zu programmieren und per JNI[1] einzubinden.

Unter Windows ist der Zugriff auf die TAP-Schnittstelle deutlich komplizierter als bei Linux. Das liegt daran, dass einige Informationen aus der Windows-Registry benötigt werden, um eine Verbindung zu dem virtuellen Netzwerkadapter herzustellen. Außerdem ist es unter Windows aufwendiger, die Netzwerkpakete zu versenden und zu empfangen. Die Quellcodes von OpenVPN[18] und QEmu[8] haben geholfen, dies zu implementieren.

Auch für den Anwender ist die Nutzung von P2PVPN unter Windows komplizierter, da er den TAP-Treiber von OpenVPN separat installiert muss. In Zukunft wäre es sinnvoll, den Treiber aus P2PVPN heraus zu installieren. Inwieweit das lizenzrechtlich und technisch möglich ist, wurde noch nicht untersucht.

Tests haben ein weiteres Problem bei Windows gefunden, das nicht direkt mit P2PVPN zusammenhängt. Manche Windowsversionen versenden UDP-Broadcast-Pakete mit einer falschen Quell-IP-Adresse. Beobachtet wurde, dass Pakete die über den virtuellen

[1] Java Native Interface

Netzwerkadapter gesendet wurden, die IP-Adresse des physikalischen Adapters hatten. Es konnte allerdings nicht herausgefunden werden, wann und warum sich Windows so verhält oder wie das Problem behoben werden kann. Damit auch UDP-Broadcasts funktionieren, überschreibt P2PVPN ggf. die Quell-IP-Adresse in UDP-Paketen mit dem richtigen Wert.

Um die IP-Adresse des virtuellen Netzwerkadapters zu setzen, kann generell ein Kommandozeilenbefehl verwendet werden. Dieser unterscheidet sich je nach Betriebssystem, kann aber in jedem Fall direkt aus Java heraus ausgeführt werden.

Andere Betriebssysteme als 32Bit-Linux und 32Bit-Windows werden derzeit nicht unterstützt. Auch wenn die Implementierung für andere Systeme leicht möglich wäre, übersteigt der Aufwand P2PVPN auf weiteren Systemen zu Kompilieren und zu Testen den Rahmen dieser Arbeit.

5.2 Das dezentrale Netzwerk und Sicherheit

5.2.1 Bouncy Castle

Java stellt in den Paketen `java.security` und `javax.crypto` Schnittstellen zu mehreren kryptografischen Algorithmen zur Verfügung. Einige Implementierungen sind bereits ein Teil von Java, aber asymmetrische Verschlüsselungsverfahren wie z. B. RSA fehlen. Allerdings lassen sich über einen *Cryptographic Service Provider* die verfügbaren Implementierungen erweitern.

Ein solcher Provider ist *Bouncy Castle* [1]. Er wird unter einer freien Lizenz veröffentlicht, sodass er von P2PVPN für die folgenden Aufgaben genutzt werden kann:

- Bouncy Castle stellt symmetrische und asymmetrische Verschlüsselungsverfahren zur Verfügung.
- Es können Signaturen erstellt und überprüft werden.
- Es werden verschiedene kryptografische Hashfunktionen bereitgestellt.
- Es können kryptografisch sichere Zufallszahlen erzeugt werden.

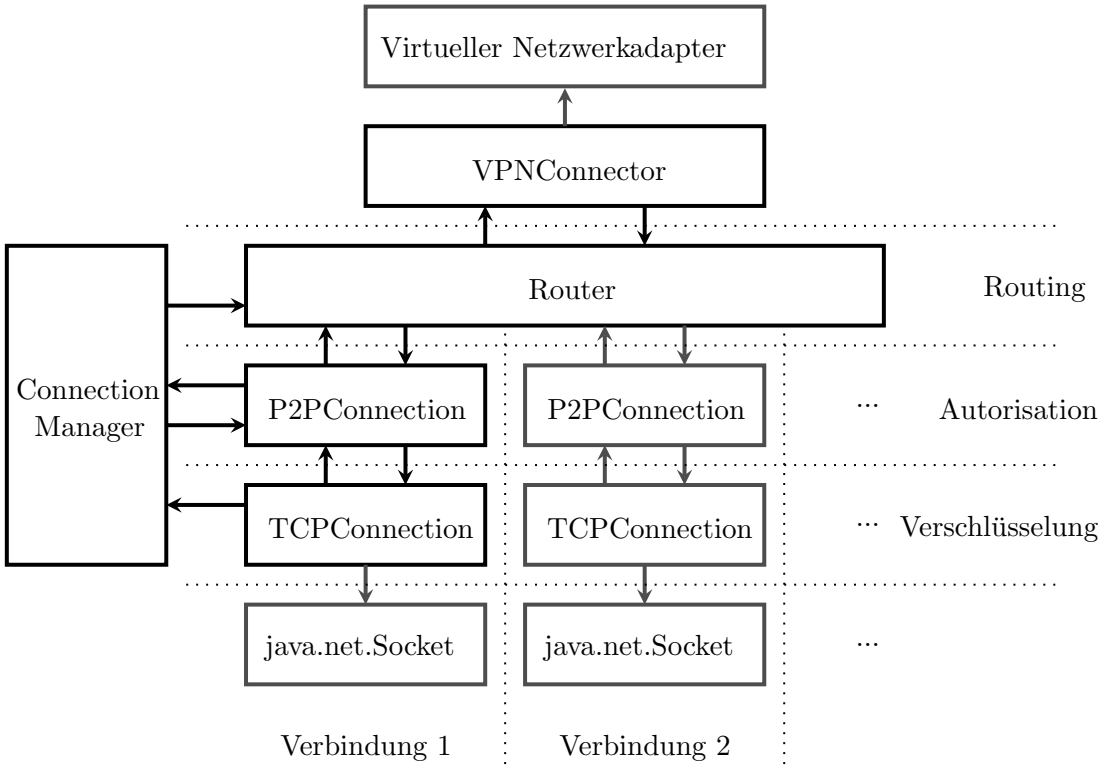


Abbildung 5.1: Entwurf der Softwarearchitektur

Nachdem der Provider dem Java Laufzeitsystem bekannt gemacht wurde, sind nur noch Aufrufe in die Java-API nötig, um den Provider zu nutzen. Die einzelnen Algorithmen werden anhand von Strings identifiziert (z. B. „`AES/CBC/ISO10126Padding`“ für AES im CBC-Modus und Padding nach ISO 10126). Diese Strings werden in P2PVPN an zentraler Stelle festgelegt, sodass sich der Provider, aber auch die Algorithmen, in Zukunft leicht ändern lassen.

5.2.2 Schichten

Die Abbildung 5.1 zeigt, wie die Schichten zusammenarbeiten. Die Kästen repräsentieren dabei Objekte. Die Pfeile zeigen, wie Objekte andere aufrufen. Der virtuelle Netzwerkadapter entspricht der Netzzugangsschicht des virtuellen Netzes. Mit der Klasse `java.net.Socket` werden TCP-Verbindungen zu anderen Knoten aufgebaut. Diese Klasse arbeitet auf der Anwendungsschicht des physikalischen Netzes. Dazwischen

liegen die drei Schichten von P2PVPN. Außerdem existiert ein Objekt der Klasse `VPNConnector`, das die Verbindung zu dem virtuellen Netzwerkadapter herstellt. Das Objekt der Klasse `ConnectionManager` verwaltet den Aufbau und die Verbindung der Schichten. Wie die Abbildung zeigt, wird für das Routing insgesamt ein Objekt benötigt. Die anderen beiden Schichten benötigen pro Verbindung ein Objekt.

Sobald eine neue ein- oder ausgehende TCP-Verbindung aufgebaut wird, wird ein neues `TCPConnection`-Objekt erzeugt, das für diese Verbindung zuständig ist. Dieses meldet sich bei dem `ConnectionManager`, welcher nun ein `P2PConnection`-Objekt erzeugt und mit der `TCPConnection` verbindet. Nun führt die `P2PConnection` die Autorisierung durch. Ist dies erfolgreich, meldet sich die `P2PConnection` beim `ConnectionManager`, welcher schließlich die Verbindung zum `Router` herstellt. Nun sind die beiden Knoten verbunden und können Pakete des VPNs austauschen.

Die Schnittstellen zwischen den Schichten sind ereignisbasiert. In den Schnittstellen zu dem virtuellen und physikalischen Netzen, d. h. in den Klassen `VPNConnection` und `TCPConnection` laufen Threads, die diese Ereignisse auslösen. Für die Kommunikation „Nach unten“ bieten die Schichten die Methoden `send` und `close` an, um Pakete zu versenden oder Verbindungen zu beenden. „Nach oben“ werden die Methoden `receive` und `connectionClosed` benutzt. Diese veranlassen die Bearbeitung eines empfangenen Paketes oder die Reaktion auf eine gerade beendete Verbindung.

Der modulare Aufbau in Schichten hat den Vorteil, dass die Protokolle der einzelnen Schichten relativ einfach sind. Alle Schichten können unterschiedliche Arten von Paketen empfangen und versenden. Bei der Routingschicht gibt das erste Byte die Art des Pakets an. Die beiden anderen Schichten brauchen ein solches Byte nicht, da sich hier die Art des Paketes aus dem Kontext erschließen lässt. Diese Schichten sind als Zustandsautomaten implementiert, die auf Ereignisse wie z. B. „ein Paket wurde empfangen“ reagieren.

5.2.3 Verschlüsselung

Da sich eine TCP-Verbindung wie ein Datenstrom verhält, wird für jedes Paket dessen Größe mitversendet. So kann der Empfänger die einzelnen Pakete auseinanderhalten. Auf der Verschlüsselungsschicht gibt es zwei Arten von Paketen, die folgendermaßen aufgebaut sind:

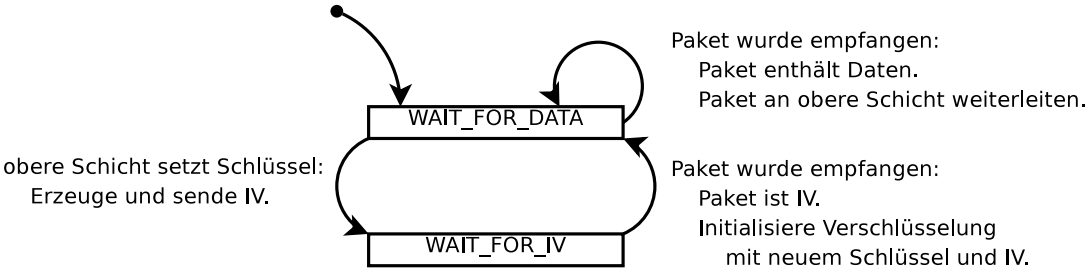


Abbildung 5.2: Zustandsautomat der Verschlüsselungsschicht

1. Initialisierungsvektor
   a) 2 Byte: Länge des Pakets
   b) n Byte: Initialisierungsvektor eventuell verschlüsselt
2. Daten
   a) 2 Byte: Länge des Pakets
   b) n Byte: Daten eventuell verschlüsselt

Die Verschlüsselungsschicht enthält einen Zustandsautomaten, der in Abbildung 5.2 abgebildet ist. Ereignisse lösen Zustandsübergänge und Aktionen aus. Der Automat entscheidet, ob ein gerade empfangenes Paket einen Initialisierungsvektor oder Daten enthält. Bevor von der oberen Schicht, also der Autorisierungsschicht, ein Schlüssel für die Verschlüsselung gesetzt wurde, werden alle Pakete unverschlüsselt übertragen. In der Praxis wird dieser Schlüssel sofort gesetzt, sodass lediglich der erste Initialisierungsvektor unverschlüsselt übertragen wird. Ändert die Autorisierungsschicht den Schlüssel, wird ein neuer Initialisierungsvektor erzeugt und übertragen. Dabei wird noch der alte Schlüssel genutzt. Die darauf folgenden Datenpakete werden mit dem neuen Schlüssel verschlüsselt.

Um Pakete zu verschlüsseln, wird der 128Bit AES im CBC-Modus und Padding nach ISO 10126 verwendet.

Um die Bandbreite beim Senden zu begrenzen, werden ggf. Pakete verworfen, die nach dem Token-Bucket-Algorithmus ermittelt werden. Tests haben gezeigt, dass der virtuelle TCP/IP-Stack dadurch die Senderate zuverlässig auf die gewünschte Bandbreite verringert. Damit auch bei mehreren Verbindungen die Bandbreite eingehalten wird, läuft der Token-Bucket-Algorithmus in einer globalen Instanz. Soll ein Paket gesendet

werden, wird unabhängig von der Verbindung bei dieser Instanz nachgefragt, ob jetzt gerade ein Paket gesendet werden darf oder nicht.

Möchte man die Empfangsraten begrenzen, ergibt sich allerdings ein Problem. Sendet die Gegenstelle zu schnell, kann man auf zwei Arten reagieren. Verwirft man Pakete, die empfangen wurden und dabei die Bandbreite überschritten haben, wird der virtuelle TCP/IP-Stack darauf reagieren und langsamer senden. Es hat sich allerdings gezeigt, dass dabei die Bandbreite deutlich unter die Begrenzung fällt.

Die andere Möglichkeit ist, nach jedem empfangenen Paket etwas zu warten. Dabei blockiert die Verbindung kurz. Die Datenflusskontrolle des physikalischen TCP/IP-Stacks wird auf diese Weise das Senden der Gegenstelle abbremsen. Hierbei wird die gewünschte Bandbreite genau erreicht. Allerdings füllen sich dadurch die Sende- und Empfangspuffer der physikalischen TCP-Verbindung, sodass eine extrem hohe Latenz entsteht.

Beide Möglichkeiten sind also nicht praktikabel. Aus diesem Grund wurde auf eine Begrenzung der Empfangsrate verzichtet.

Um die Bandbreite zu begrenzen, die dadurch entsteht, dass Pakete weitergeleitet werden, reicht es die Senderate einzuschränken. Außerdem haben die meisten Internetanschlüsse eine hohe Empfangsbandbreite und eine niedrige Sendebandbreite. Die Empfangsbandbreite stellt normalerweise keine Ressource da, mit der sparsam umgegangen werden muss.

5.2.4 Autorisierung

Damit Einladungen vom Benutzer leicht per E-Mail oder auf anderem Wege verschickt werden können, sollte sie in ASCII vorliegen. Hierfür wurde eine Erweiterung von `java.utils.Properties` implementiert. Sie heißt `AdvProperties` und kann genauso wie die Basisklasse Schlüssel/Wert-Paare speichern und als ASCII-Text ausgeben oder einlesen. Außerdem ist das Folgende möglich:

- Als Werte können `byte`-Arrays gespeichert werden. Diese werden Base64 codiert und ggf. auf mehrere Schlüssel verteilt, um die Zeilenlänge gering zu halten.
- Bei dem Speichern werden die Schlüssel sortiert ausgegeben. Außerdem werden die sonst üblichen Kommentare Dateikopf entfernt.

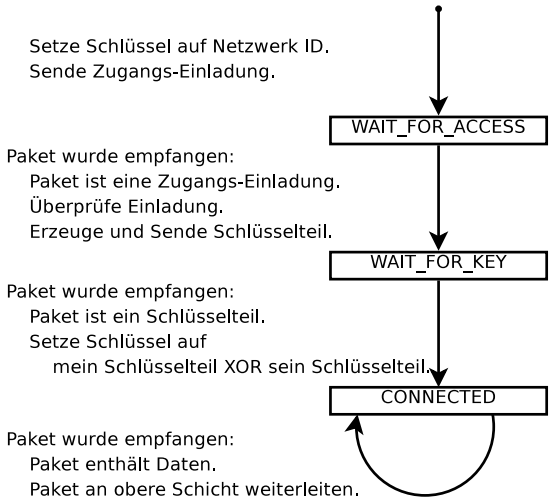


Abbildung 5.3: Zustandsautomat der Autorisierungsschicht

- `AdvProperties` können signiert werden. Die Signatur wird als `byte`-Array in einem bestimmten Schlüssel/Wert-Paar abgelegt.

Die Autorisierungsschicht verwendet drei Arten von Paketen:

1. Zugangs-Einladung
   a) Die Zugangs-Einladung als ein langer, von `AdvProperties` erzeugter ASCII String.
2. Schlüsselteil
   a) 128 Bit zufällige Daten, die als Schlüssel verwendet werden.
3. Daten
   a) Ein Datenpaket, dass an die Routingschicht weitergegeben wird.

Der in Abbildung 5.3 zu sehende Automat entscheidet, von welchem Typ die empfangenen Pakete sind. Außerdem führt er das in Kapitel 4.3.2 beschriebene Protokoll aus.

5.2.5 Routing

Die Routingschicht hat die Aufgabe, die verteilte Datenbank aktuell zu halten und Pakete an die richtigen Knoten weiterzuleiten. Außerdem bietet die Routingschicht

die Möglichkeit, P2PVPN-interne Pakete zu versenden, die nicht durch den virtuellen TCP/IP-Stack laufen. Insgesamt werden dafür fünf verschiedene Paketarten benötigt:

1. Datenpaket
    a) 1 Byte: Typ = `DATA_PAKET`
    b) n Byte: Ethernet Paket (Zieladresse steht im Ethernet-Kopf)
2. Broadcast-Datenpaket
    a) 1 Byte: Typ = `DATA_BROADCAST_PACKET`
    b) 6 Byte: Zieladresse
    c) n Byte: Ethernet Paket
3. Datenbankanfrage
    a) 1 Byte: Typ = `ASK_DB`
    b) n Byte: Serialisierte `PeerID`
    c) m Byte: Serialisierte lokale Datenbankversion (`long`)
4. Datenbankantwort
    a) 1 Byte: Typ = `SEND_DB`
    b) n Byte: Serialisierte `PeerID`
    c) m Byte: Serialisierte Datenbank (`VersionizedMap`)
5. Internes Paket
    a) 1 Byte: Typ = `INTERNAL_PAKET`
    b) 1 Byte: Subtyp
    c) 6 Byte: Zieladresse
    d) n Byte: Paketdaten

Die Abbildung 5.4 zeigt den Aufbau eines normalen Datenpakets mit allen relevanten Schichten. Mit diesen Paketen wird nahezu der gesamte Datenverkehr übertragen.

Wie im Entwurf beschrieben, müssen Broadcast-Pakete von P2PVPN einzeln an jedes erreichbare Ziel verschickt werden. Damit diese Pakete eindeutig adressiert werden können, muss die jeweilige Zieladresse im Paket enthalten sein. Hierfür wird das zusätzliche Adressfeld genutzt.

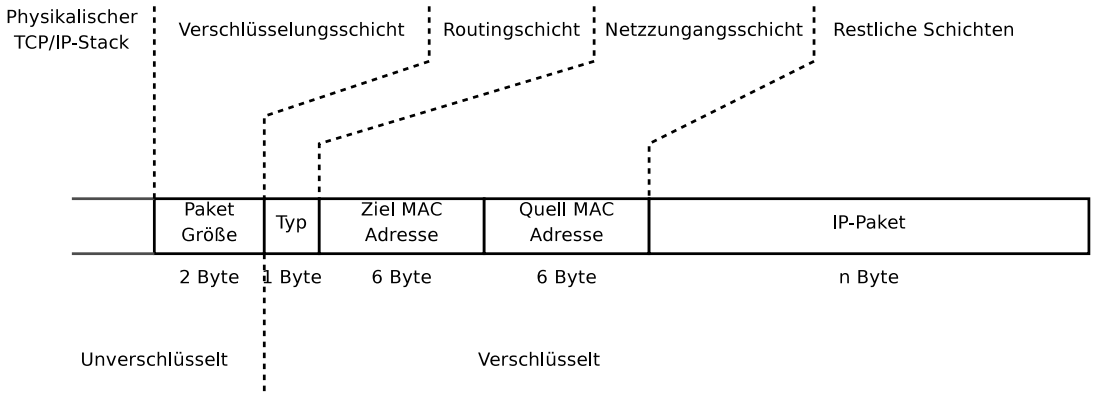


Abbildung 5.4: Inhalt eines normalen Datenpakets

Um die Datenbank zu synchronisieren, wird die Serialisierung von Java genutzt. Jeder Knoten sendet wie in Kapitel 4.2.2 beschrieben Datenbankanfragen an seine Nachbarn. Diese antworten ggf. mit einer Datenbankantwort. Da diese Pakete immer nur an Nachbarn gesendet werden, ist es nicht nötig, dass sie eine Zieladresse enthalten.

Um die Versionsnummern zu verwalten, wurde die Klasse `VersionizedMap` implementiert, die sich von `java.util.HashMap` ableitet. `VersionizedMap` hat einen Zähler, der die Anzahl der Änderungen zählt. Dieser Zähler wird als Versionsnummer verwendet.

Interne Pakete werden momentan verwendet, um die Latenz zu Nachbarn zu ermitteln, und um Chat-Nachrichten zu übertragen. Hierfür sind insgesamt drei Subtypen von Paketen nötig:

1. Ping-Anfrage
    a) 1 Byte: Typ = `INTERNAL_PAKET`
    b) 1 Byte: Subtyp = `INTERNAL_PORT_PING`
    c) 6 Byte: Zieladresse
    d) 1 Byte: Ping-Typ = `PING_REQUEST`
    e) 2 Byte: Ping-Paket ID
    f) 6 Byte: Lokale (sender) Adresse
2. Ping-Antwort
    a) 1 Byte: Typ = `INTERNAL_PAKET`
    b) 1 Byte: Subtyp = `INTERNAL_PORT_PING`

   c) 6 Byte: Zieladresse
   d) 1 Byte: Ping-Typ = `PING_REPLY`
   e) 2 Byte: Ping-Paket ID

3. Chat-Paket

   a) 1 Byte: Typ = `INTERNAL_PAKET`
   b) 1 Byte: Subtyp = `INTERNAL_PORT_CHAT`
   c) 6 Byte: Zieladresse
   d) n Byte: Chat-Nachricht in UTF-8

Möchte der Knoten A die Latenz zu einem Knoten B ermitteln, sendet er ein Paket zu B, das eine eindeutige zufällige Nummer und seine Adresse enthält. Der Knoten B antwortet unmittelbar mit der gleichen eindeutigen Nummer. Der Knoten A kann anhand der Nummer das empfangene Paket dem gesendeten zuordnen und die Latenz berechnen.

Das Chat-Paket wird für einen sehr einfachen Chat benötigt. Dieser Chat ermöglicht es Benutzern, Nachrichten an alle Teilnehmer im gleichen P2PVPN Netz zu senden.

5.3 Grafische Benutzeroberfläche

Ein wichtiges Ziel von P2PVPN ist die Benutzerfreundlichkeit. Deswegen sind alle Konfigurationsmöglichkeiten oder Aktionen, die ein Benutzer durchführen kann, über eine grafische Oberfläche möglich. Dabei wird darauf geachtet, dass niemand bei der Benutzung von P2PVPN zu sehr mit dessen Komplexität konfrontiert wird. Trotzdem gibt es Einstellungsmöglichkeiten, die ein Verständnis von TCP/IP und Netzwerken voraussetzen. Diese werden aber automatisch mit vernünftigen Voreinstellungen belegt.

Der Aufbau der Oberfläche berücksichtigt vor allem drei Aufgaben, die ein Benutzer möglichst intuitiv durchführen können soll:

- Ein neues Netzwerk konfigurieren und erstellen.
- Eine Einladung erzeugen.
- Einem Netzwerk beitreten.

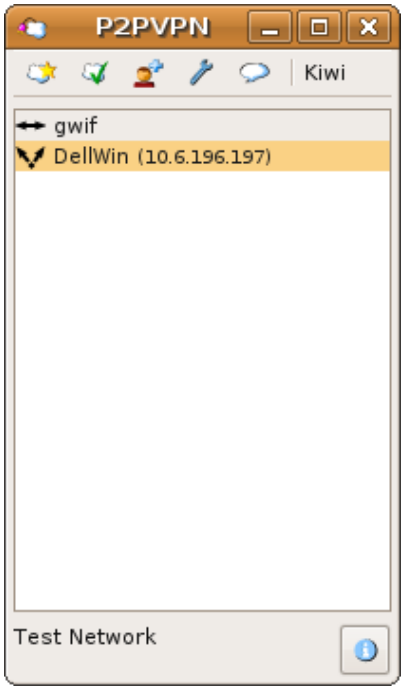


Abbildung 5.5: Das Hauptfenster von P2PVPN

Das Hauptfenster (Abbildung 5.5) von P2PVPN besteht überwiegend aus einer Liste aller Knoten im Netzwerk. Über dieser Liste steht der Name des Knotens, und unter ihr der Name des Netzwerks. Dies ist also eine Übersicht über die wichtigsten Informationen. Außerdem gibt es sechs Knöpfe, die weitere Fenster öffnen und verschiedene Aktionen ermöglichen. Diese Knöpfe enthalten Piktogramme, um klein und übersichtlich zu bleiben. Außerdem beschreiben Tooltips die genauere Bedeutung. Im Folgenden werden diese Knöpfe genauer beschrieben.

5.3.1 Ein neues Netzwerk erstellen

Mit kann ein neues Netzwerk erstellt werden. Dabei sind diese Einstellungen möglich:

Network Name ist der Name des Netzwerks.

Network ist der IP-Adressbereich des virtuellen Netzwerks. (Voreinstellung: 10.6.0.0)

Subnet Mask ist die Subnetzmaske, (Voreinstellung: 255.255.0.0)

Known Hosts ist eine Liste mit bekannten IP-Adressen oder DNS-Namen, die für das Bootstrapping verwendet wird.

BitTorrent Tracker gibt den BitTorrent-Tracker an, der für das Bootstrapping verwendet wird. (Voreinstellung: `http://tracker.thepiratebay.org:80/announce`)

Keine dieser Einstellungen muss verändert werden, um ein funktionierendes Netzwerk zu erzeugen. Nach dem Erzeugen eines Netzes tritt P2PVPN diesem automatisch bei. Außerdem wird für diesen Knoten eine zufällige IP-Adresse gewählt, die in dem angegebenen Adressbereich liegt.

5.3.2 Einem bestehenden Netzwerk beitreten

Hat ein Benutzer eine Einladung bekommen und möchte dieser nachkommen, muss er wählen. In dem Fenster, das sich dann öffnet, kann eine Einladung aus einer Datei geladen werden oder per Cut&Paste eingefügt werden. Es wird automatisch erkannt, ob es sich um eine Netzwerk- oder Zugangs-Einladung handelt. Intern wird ggf. eine Zugangs-Einladung generiert, wenn der Benutzer eine Netzwerk-Einladung bekommen hat. Auch hier wird eine zufällige IP-Adresse erzeugt.

5.3.3 Eine Einladung erstellen

Über kann eine Einladung erstellt werden. Diese Aktion ist nur verfügbar, wenn die Netzwerk-Einladung des aktuellen Netzes bekannt ist. Der Benutzer muss wählen, ob er eine Zugangs- oder Netzwerk-Einladung erstellen will. Für eine Zugangs-Einladung kann hier auch das Verfallsdatum angegeben werden. Diese kann in einer Datei gespeichert werden oder per Cut&Paste in eine andere Anwendung kopiert werden.

5.3.4 Die Einstellungen ändern

In dem Einstellungsfenster können diese Einstellungen vorgenommen werden:

Your Name ist der Name dieses Knotens.

Listen on Port gibt den physikalischen Port an, an dem Verbindungen von anderen Knoten entgegen genommen werden. (Voreinstellung: lasse das Betriebssystem einen Port wählen)

Max Upload ist die maximale Sendebandbreite. (Voreinstellung: keine Begrenzung)

Popup chat on incoming message gibt an, ob der Benutzer durch einen Popup auf eine neue Chat-Nachricht aufmerksam gemacht werden soll. (Voreinstellung: Nein)

VPN IP ist die virtuelle IP-Adresse des Knotens. (Wird automatisch sinnvoll belegt)

Send buffer size ist die Größe des Puffers in der Verschlüsselungsschicht. (Voreinstellung: 10)

Auch diese Einstellungen müssen nicht verändern werden, damit P2PVPN funktioniert. Zusätzlich zu einer Änderung des Namens werden viele Anwender den Port setzen. Dies ist nötig, um in einem NAT-Router eine Port-Weiterleitung einzurichten.

Die Puffergröße hat Einfluss auf den Datendurchsatz, der mit P2PVPN zu erreichten ist. Ein kleiner Wert hat die Folge, dass die virtuellen Pakete nicht effizient in physikalische Pakete verpackt werden können. Ein großes virtuelles Paket wird so zwei physikalische Pakete benötigen. Eine große Puffergröße hingegen erhöht die Latenz.

Dieser doch sehr technische Wert wurde in die Optionen aufgenommen, um es anderen Benutzern zu ermöglichen zu experimentieren. Eigene Untersuchungen und die Rückmeldungen Anderer haben ergeben, dass der Wert 10 ein Optimum darstellt.

5.3.5 Chat

 öffnet einen einfachen Chat. Jede Nachricht wird hier an alle Teilnehmer des Netzes gesendet. Der Chat hat sich als nützlich erwiesen, unbekannte Personen anzusprechen, um sie nach Erfahrungen und Problemen mit P2PVPN zu befragen.

5.3.6 Weitere Informationen

Mit  wird ein Fenster geöffnet, dass weitere Informationen anzeigt. Diese Informationen sind bei dem normalen Betrieb nicht wichtig, aber vor allem bei der Fehlersuche interessant. Das Folgende ist hier zu sehen:

- Der Inhalt der verteilten Datenbank
- Die Latenz und die genutzte Bandbreite zu benachbarten Knoten
- Physikalische IP-Adressen unter denen andere Knoten erreichbar sein könnten
- Lognachrichten

6 Ergebnisse

6.1 Überblick über VPN Software

Es gibt unzählige Softwareprodukte, die ein virtuelles privates Netzwerk aufbauen. Die große Mehrzahl von diesen verwendet allerdings eine Client-Server-Topologie. *OpenVPN* ist hier wohl einer der bekanntesten Vertreter. Eine dezentrale Topologie haben nur wenige Programme, wie z. B. *Hamachi*, *Wippien* oder *n2n*.

Diese Programme werden hier kurz beschrieben und mit P2PVPN verglichen. Tabelle 6.1 zeigt, welche Programme eine grafische Benutzeroberfläche haben und ob eine dezentrale Netzwerktopologie verwendet wird. Beides ist für die Benutzerfreundlichkeit wichtig. Außerdem ist angegeben, ob eine freie Lizenz verwendet wird. Durch eine freie Lizenz ist man nicht gezwungen, dem Urheber zu vertrauen.

6.1.1 OpenVPN

OpenVPN[18] wird von der Firma OpenVPN Technologies entwickelt, die ihren Sitz in den USA hat. OpenVPN wird dabei unter einer freien Lizenz vertrieben.

OpenVPN kann sehr umfangreich und flexibel über Konfigurationsdateien eingerichtet werden. Um diese Dateien zu erstellen, gibt es diverse grafische Oberflächen. Besonders auf der Client-Seite lässt sich OpenVPN so auch ohne große Kenntnisse einrichten. Wie bei jeder Software mit einer Client-Server-Topologie muss allerdings auch ein Server konfiguriert und betrieben werden, was den Nutzen für Privatpersonen einschränkt.

Um Pakete zu transportieren kann TCP oder UDP genutzt werden. Die Daten werden mit SSL verschlüsselt.

	OpenVPN	Hamachi	Wippien	n2n	P2PVPN
GUI	Ja, durch Drittanbieter	Nur Windows	Nur Windows	Nein	Ja
Topologie	Client-Server	Dezentral	Dezentral	Dezentral	Dezentral
Lizenz	Frei	Proprietär	Teilweise Frei	Frei	Frei
Plattformen	Linux, Mac OS X, Windows etc.	Linux, Mac OS X, Windows	Linux, Windows	Linux, Mac OS X, Windows etc.	Linux, Windows
Version	2.0.9	1.0.3.0 und 0.9.9.9-20	2.2.7	1.3.2	0.7

Tabelle 6.1: VPN-Software im Vergleich

6.1.2 Hamachi

Hamachi[16], aber auch Wippien sind P2PVPN am ähnlichsten und es hat sich gezeigt, dass viele Nutzer von P2PVPN auf der Suche nach einer Alternative zu diesen beiden Programmen waren. Das proprietäre Hamachi wird von der Firma LogMeIn aus Kanada entwickelt. Es gibt eine kostenlose Version, die allerdings nur für nicht kommerzielle Zwecke genutzt werden darf. Wird Hamachi für kommerzielle Zwecke genutzt, muss ein monatlicher Betrag gezahlt werden.

Für Windows existiert grafische Oberfläche. Linux und Mac OS X Nutzer müssen mit einer einfachen Kommandozeilenversion vorlieb nehmen.

Wenn möglich, bauen die einzelnen Hamachi-Knoten eines Netzes direkte Verbindungen zueinander auf. Ist eine direkte Verbindung nicht möglich, wird der Datenverkehr durch Server von LogMeIn weitergeleitet. Außerdem wird der Verbindungsaufbau zwischen den Knoten von LogMeIn koordiniert.

Auf diese Weise ist es sehr leicht, Hamachi zu konfigurieren und zu nutzen. Wie ein Artikel in der c´t richtig feststellt, hat dies aber auch seine Nachteile:

> Bei aller Bequemlichkeit muss man sich deshalb bewusst sein, dass man mit der Nutzung ein Vertrauensverhältnis mit dem Betreiber eingeht. Denn der Quellcode der Client-Tools ist nicht offen gelegt.[6]

Vermutlich wurde nicht zuletzt wegen solcher Kritiken das Netzwerkprotokoll von Hamachi veröffentlicht[15]. Dies zeigt, wie verhindert wird, dass der Datenverkehr zwischen den Knoten durch Dritte oder LogMeIn abgehört werden kann. Ob die Implementierung diesem Protokoll entspricht, ist allerdings nicht überprüfbar. Aber auch wenn dies der Fall ist, weiß LogMeIn welche ihrer Kunden sich zusammen in welchen Netzwerken aufhalten. Außerdem kennt LogMeIn die Zugangsdaten zu allen Netzwerken, sodass sie ggf. an diesen teilnehmen können.

6.1.3 Wippien

Wippien[19] ist eine VPN Software, die auf dem Instant-Messaging-Protokoll Jabber aufbaut. Wippien verhält sich wie ein Jabber-Client, der es zusätzlich erlaubt, VPN-Verbindungen zu anderen Nutzern aufzubauen.

Wippien wird von einer Privatperson aus Kroatien entwickelt. Allerdings ist die Software nicht, wie der erste Eindruck der Webseite suggeriert, komplett quelloffen. Wichtige Kernkomponenten sind proprietär und stammen von der kroatischen Firma *WeOnlyDo! Software*. Z. B. der virtuelle Netzwerkadapter und die Implementierung des dezentralen Netzes sind komplett proprietär. Das verwendete Netzwerkprotokoll ist nicht bekannt.

Allerdings muss man Wippien zugutehalten, dass die Koordinierung des Verbindungsaufbaus über das dezentrale Jabber geschieht. So gibt es keine zentrale Stelle, die die Teilnehmer der verschiedenen Netze kennt.

Für Windows gibt es eine grafische Oberfläche. Eine Kommandozeilenversion für Linux ist auch verfügbar. Allerdings kann die Linuxversion nur genutzt werden, wenn zuvor mit der Windowsversion konfiguriert wurde, welche Personen eine VPN-Verbindung zu dem lokalen Knoten aufbauen dürfen.

6.1.4 n2n

n2n[2] ist eine kommandozeilenbasierte dezentrale VPN Software, die als Open-Source-Projekt entwickelt wird. Damit ist es neben P2PVPN eines der wenigen freien und dezentralen Lösungen.

n2n kennt zwei verschiedene Arten von Knoten. *Edge Nodes* laufen auf den Rechnern, die an dem virtuellen Netzwerk teilnehmen. Damit Edge Nodes sich finden und Verbindungen zueinander aufbauen können brauchen sie die Hilfe von *Super Nodes*. Diese können ggf. auch Daten weiterleiten, falls eine Verbindung zwischen zwei Edge Nodes nicht möglich ist. Super Nodes müssen dabei immer unter einer festen Adresse erreichbar sein.

n2n ist für diverse Plattformen verfügbar, wobei eine kompilierte Version für Windows käuflich erworben werden muss. Inwieweit es möglich ist, aus den Quellen eine Windowsversion zu kompilieren, wurde nicht untersucht.

6.1.5 P2PVPN

Alle oben beschriebenen Programme brauchen zum Finden anderer Knoten und zum Weiterleiten von Daten eine übergeordnete Struktur. P2PVPN besitzt hierfür keine Hierarchie, da jeder Knoten ggf. diese Aufgaben übernehmen kann. Dadurch muss eine solche Struktur weder aufgebaut noch ihr vertraut werden. Eine Ausnahme bildet hier die Nutzung eines BitTorrent-Trackers. Die Informationen, die dieser erhält, sind allerdings so gering wie möglich.

Auch die grafische Oberfläche von P2PVPN ist ein Alleinstellungsmerkmal. Im Rahmen dieser Arbeit konnte keine dezentrale VPN-Software gefunden werden, die unter einer freien Lizenz veröffentlicht wird und eine grafische Oberfläche hat. Vor allem unter Linux sind grafische Oberflächen rar.

6.2 Leistung

6.2.1 Routing

Da der Routingalgorithmus darauf basiert, dass jeder Knoten die gesamte Netzstruktur kennt, hat er eine vergleichsweise hohe Komplexität. In einem Netz mit n Knoten kann es maximal $\frac{n^2-1}{2}$ Verbindungen geben. Die verteilte Datenbank hat also eine Größe von $O(n^2)$, die übertragen und verarbeitet werden muss. Daraus ergeben sich die folgenden Komplexitäten. Sie geben den schlechtesten Fall an und beziehen sich immer auf nur einen Knoten:

Speicherbedarf :	$O(n^2)$
Übertragene Datenmenge bei einer Änderung:	$O(n^2)$
Berechnen einer Route:	$O(n^2)$
Berechnen aller Routen:	$O(n^3)$
Route im Cache nachschlagen:	fast $O(1)$

Der Speicherbedarf teilt sich in n Teile auf, die jeweils ein Knoten ändern kann. Wenn alle Knoten ihren Teil ändern, muss jeder Knoten die komplette Datenbank aktualisieren. Es müssen also $O(n^2)$ an Daten empfangen werden. Die gleiche Menge an Daten muss auch gesendet werden.

Um eine Route zu finden, muss der kürzeste Weg in einem Graphen gefunden werden. Da alle Kantengewichte gleich sind, eignet sich hier eine Breitensuche. Die Laufzeit einer Breitensuche ist proportional zu der Anzahl der Knoten und der Kanten. Daraus ergibt sie die Laufzeit von $O(n^2)$ um eine Route zu berechnen. Da in einem (virtuellen) Ethernet-Netzwerk regelmäßig Broadcasts verschickt werden, benötigt jeder Knoten die Routen zu allen anderen. Nach einer Änderung in der Netzwerktopologie braucht jeder Knoten also eine Rechenzeit von $O(n^3)$.

Berechnete Routen werden in einem Cache gespeichert. Für jedes Paket, das gesendet wird, muss in diesem Cache nachgeschlagen werden. Als Cache wird eine Hash-Tabelle von Java genutzt. Bei einer effizienten Implementierung ist das Lesen eines Wertes aus einer Hash-Tabelle mit fast $O(1)$ möglich.

Es ist also deutlich, dass sich der implementierte Routingalgorithmus nicht für große Netze eignet. Aber schon die Tatsache, dass es sich um ein Ehternet-Subnetz handelt, begrenzt die Größe. Es hat sich gezeigt, dass die übliche Knotenanzahl in einem P2PVPN-Netz nicht durch den Routingalgorithmus begrenzt wird.

6.2.2 Netzwerk

Wie jede VPN-Software verringert P2PVPN die mögliche Bandbreite und erhöht die Latenz. Jedes gesendete Paket muss zwei TCP/IP-Stacks und die VPN-Software durchlaufen. Das kostet Zeit und erhöht den Overhead.

Bandbreite

Unter der Annahme, dass bei einer Datenübertragung alle Pakete voll ausgefüllt sind, lässt sich der Verlust an Bandbreite folgendermaßen berechnen: Wie schon in Kapitel 4.2.1 beschrieben, kann ein TCP-Paket 1460 Bytes an Nutzdaten enthalten. Ein solches Paket hat auf der Netzzugangsschicht eine Größe von 1514 Bytes. Daraus ergibt sich, dass für jedes Nutzbyte im Mittel $3,7\%$ Verwaltungsdaten hinzukommen.

An die 1514 Bytes hängt die Routingschicht von P2PVPN 1 Byte. Durch die Blockchiffre der Verschlüsselungsschicht wird diese Paketgröße auf einen durch 16 teilbaren Wert aufgerundet. Dazu kommen die 2 Bytes, die die Paketgröße angeben. Es ergeben sich 1522 Bytes. Das sind 0.5% an zusätzlichen Verwaltungsdaten.

Verglichen mit einer direkten Übertragung über TCP/IP, werden beim Nutzen von P2PVPN $4,2\%$ mehr an Verwaltungsdaten übertragen. Anders gesagt braucht die Übertragung einer bestimmten Datenmenge $4,2\%$ länger, wenn sie durch P2PVPN läuft.

Messungen haben ergeben, dass P2PVPN in der Praxis um etwa 10% langsamer ist als eine direkte Verbindung über das Internet. Dies könnte daran liegen, dass P2PVPN die Flusssteuerung von TCP/IP stört, sodass die Daten nicht so schnell gesendet werden, wie es möglich wäre.

Latenz

Die Latenz wird durch ein VPN erhöht, da mehr Software ausgeführt werden muss, um ein Paket zu verarbeiten. In einem LAN ist eine Latenz von $0,2$–$0,5ms$ üblich. Lässt man P2PVPN in einem LAN laufen, ist die Latenz im virtuellen Netz etwa $1,5ms$. Ein Paket braucht also ca. $1ms$ um P2PVPN vier Mal zu durchlaufen.

6.2.3 Prozessorauslastung in der Praxis

Um VPN-Software zu implementieren, wird in der Regel C oder C++ verwendet. So können betriebssystemnahe und effiziente Programme erstellt werden. Java hingegen hat den Ruf, langsam zu sein. Die Tabelle 6.2 zeigt, wie stark der Prozessor die Bandbreite von P2PVPN begrenzt. In beiden Fällen wurde Java 1.6 unter Linux verwendet.

Rechner	Maximale Bandbreite (MB/s)
Intel Core Duo 1.73GHz	3,2
Intel Celeron M 900MHz	1,7

Tabelle 6.2: Begrenzung der Bandbreite durch die Prozessorleistung

Wie zu sehen ist, begrenzt P2PVPN die Bandbreite in einem LAN. Eine andere Programmiersprache oder eine effizientere Implementierung könnten diese Begrenzung aufheben. Für den Betrieb über das Internet eignet sich Java aber ohne weiteres.

6.3 Weitere Schritte

6.3.1 Routing

Der Routingalgorithmus von P2PVPN ist für viele Anwendungsfälle ausreichend, hat aber zwei Schwächen:

- Der Routingalgorithmus skaliert schlecht und ist nur für kleine Netze geeignet.
- Gibt es mehrere Knoten, die Daten weiterleiten können, wird die Last nicht optimal verteilt.

Eine große Komplexität hat das Aktualisieren der verteilten Datenbank mit $O(n^2)$. Dies liegt unter anderem daran, dass jeder Knoten die gesamte Datenbank kennen muss. Grundlegende Änderungen wären nötig, um das zu ändern.

Einfacher ist es, das Protokoll zur Synchronisierung der Datenbank zu optimieren. Momentan werden immer komplette Teile der Datenbank übertragen, die eine Größe von $O(n)$ haben. Würden nur Änderungen übertragen, reichten $O(1)$. Bei einer Änderung im Verbindungsgraphen müsste jeder Knoten so nur $O(n)$ übertragen. Damit würde das Routing das Netz nicht mehr belasten als ein Broadcast.

Um die Last auf mehrere Knoten zu verteilen, wird die Latenz als Maß für die Last verwendet. Wie in Kapitel 4.2.3 beschrieben ist die Latenz alleine dafür allerdings nicht ausreichend. Wie eine bessere Lastverteilung implementiert werden kann, sollte ggf. untersucht werden.

6.3.2 NAT-Traversal

Es hat sich gezeigt, dass viele Nutzer ihren NAT-Router nicht so konfigurieren, dass P2PVPN Verbindungen von außen entgegen nehmen kann. Daraus folgt, dass Daten von Koten weitergeleitet werden, was vor allem die Latenz des virtuellen Netzes senkt. Wie schon im Kapitel 3.2 beschrieben, ist es deswegen angebracht, dass P2PVPN NAT-Router durchdringen kann.

STUN[3] ist ein Verfahren, das dies recht zuverlässig ermöglicht. Allerdings basiert es auf UDP. Da P2PVPN ausschließlich mit Paketen arbeitet, könnte auch UDP prinzipiell zum Transport genutzt werden. Allerdings ist die Übertragung bei UDP nicht gesichert. D. h., Pakete können ohne weiteres verlohren gehen. Die Autorisierungsschicht und die Verschlüsselungsschicht benötigen in der aktuellen Implementierung hingegen eine gesicherte Übertragung. Diese Schichten müssten für UDP also angepasst werden.

Außerdem benötigt STUN einen STUN-Server, also einen zentralen Dienst, der für P2PVPN grundsätzlich vermieden werden soll. Inwieweit dieser eventuell vermieden werden kann, müssten weitere Untersuchungen zeigen.

6.3.3 Grafische Oberfläche

Einige Anwender vermissen in P2PVPN die Möglichkeit, anderen Benutzern private Nachrichten schreiben zu können. Der aktuell implementierte Chat übermittelt alle Nachrichten an jeden. Dieser ließe sich leicht so erweitern, dass bestimmte Nachrichten nur an einzelne Benutzer gesendet werden.

Ein weiter Punkt, der bemängelt wurde, ist, dass P2PVPN momentan nur an einem Netz teilnehmen kann. Es wäre hilfreich, wenn mehrere Netze konfiguriert und gespeichert werden können. In einem ersten Schritt sollte es möglich sein, aus diesen Netzen ein aktives auszuwählen. Mehrere gleichzeitig aktive Netze könnten in einem zweiten Schritt möglich sein.

6.3.4 Nutzergemeinschaft

Trotz zurückhaltender Werbung ist P2PVPN auf unerwartetes Interesse gestoßen. Suchanfragen im Internet nach P2PVPN zeigen mehrere Foren und Wikis, in denen

das Programm erwähnt oder diskutiert wird. Es melden sich regelmäßig Personen mit Fragen oder Fehlerberichten. Außerdem gibt es eine Bereitschaft, Zeit zu investieren, um das Projekt zu unterstützen.

Unterstützung findet das Projekt zurzeit vor allem durch Testen und Werbung machen. Beides hat sich als sehr hilfreich erwiesen. Außerdem ist ein Entwickler gerade dabei, P2PVPN auf eine MIPS-Plattform zu portieren. Durch eine bessere Organisation der Mitarbeit wären sicherlich noch deutlich mehr Personen bereit zu helfen. Die erfolgreiche Führung von Open-Source-Projekten wird unter anderem in [11] beschrieben.

6.4 Fazit

Diese Arbeit zeigt, wie kleine dezentrale Netze aufgebaut werden können. Diese Technologie wurde genutzt, um ein virtuelles privates Netz zu implementieren. Sicherheit und Benutzerfreundlichkeit stehen sich oft entgegen. Die dezentrale Topologie ermöglicht hier aber beides.

Die Softwarearchitektur wurde so entworfen, dass sie leicht erweitert und verändert werden kann. Die verteilte Datenbank und die internen Pakete können sehr flexibel eingesetzt werden. Außerdem wird es in Zukunft möglich sein, UDP anstelle von TCP zu benutzen, um Pakete zu transportieren. So wird es möglich sein, viele NAT-Router zu durchdringen.

Die vielfältigen Konfigurationsmöglichkeiten, die bei VPN-Software üblich sind, wurden bewusst vermieden. P2PVPN konfiguriert die Netzwerkeinstellungen des virtuellen Netzes weitgehend automatisch, um einer möglichst großen Nutzergruppe zugänglich zu sein.

Der Aufbau von P2PVPN macht es gewollt unmöglich, die Verbreitung oder die Anwendungsbereiche der Software nachzuvollziehen. Die gut 2500 Downloads der Software und die Rückmeldungen von Benutzern vor allem aus Europa zeigen aber, dass P2PVPN für viele interessant ist.

Literaturverzeichnis

[1] *The Legion of the Bouncy Castle Java Cryptography APIs.* – URL `http://www.bouncycastle.org/java.html`. – [Online; Stand 4. Juni 2009]

[2] *n2n: a Layer Two Peer-to-Peer VPN.* – URL `http://www.ntop.org/n2n/`. – [Online; Stand 22. Juni 2009]

[3] *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs).* 2003. – URL `http://tools.ietf.org/html/rfc3489`. – [Online; Stand 28. Januar 2009]

[4] *IEEE Std 802.3-2005, Section One.* 2005. – URL `http://standards.ieee.org/getieee802/802.3.html`. – [Online; Stand 26. Mai 2009]

[5] *The Transport Layer Security (TLS) Protocol Version 1.2.* 2008. – URL `http://tools.ietf.org/rfcmarkup/5246`. – [Online; Stand 8. Juli 2009]

[6] Ahlers, Ernst: Das Netz im Netz. In: *c't* (2006), Juli. – URL `http://www.heise.de/ct/Ein-eigenes-LAN-sicher-durchs-Internet-spannen--/artikel/125809`

[7] Apple Inc.: *Networking - Bonjour.* 2009. – URL `http://developer.apple.com/networking/bonjour/index.html`. – [Online; Stand 11. Juli 2009]

[8] Bellard, Fabrice: *QEMU open source processor emulator.* 2009. – URL `http://www.qemu.org/`. – [Online; Stand 6. Juli 2009]

[9] Buford, John ; Yu, Heather ; Lua, Eng K.: *P2P Networking and Applications.* Morgan Kaufmann, 2008

[10] Cohen, Bram: *The BitTorrent Protocol Specification.* 2008. – URL `http://www.bittorrent.org/beps/bep_0003.html`. – [Online; Stand 2. Juli 2009]

[11] Fogel, Karl: *Producing Open Source Software*. O'REILLY, 2005. – URL `http://producingoss.com`. – [Online; Stand 10. Juli 2009]

[12] Krasnyansky, Maxim: *VTun - Virtual Tunnels over TCP/IP networks*. 2007. – URL `http://vtun.sourceforge.net/`. – [Online; Stand 6. Juli 2009]

[13] Krasnyansky, Maxim ; Yevmenkin, Maksim: *Universal TUN/TAP device driver*. 2000. – URL `http://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt`. – [Online; Stand 18. Februar 2009]

[14] Kurose, James F. ; Ross, Keith W.: *Computer Networking: A Top-Down Approach*. Addison Wesley, 2008

[15] LogMeIn Inc.: *LogMeIn Hamachi - Security Architecture*. – URL `https://secure.logmein.com/products/hamachi/securityarchitecture.asp`. – [Online; Stand 23. Januar 2009]

[16] LogMeIn Inc.: *VPN Evolved: Sicherer Remotezugriff mit LogMeIn Hamachi*. – URL `http://www.logmeinhamachi.com/`. – [Online; Stand 23. Januar 2009]

[17] Matthes, Roland: *Algebra, Kryptologie und Kodierungstheorie*. Fachbuchverlag Leipzig, 2003

[18] OpenVPN Technologies, Inc.: *Welcome to OpenVPN*. – URL `http://www.openvpn.net/`. – [Online; Stand 23. Januar 2009]

[19] Petric, Kresimir: *Free P2P VPN software - Wippien*. – URL `http://wippien.com/`. – [Online; Stand 22. Juni 2009]

[20] Plössl, Klaus: *Mehrseitig sichere Ad-hoc-Vernetzung von Fahrzeugen*. Gabler Verlag, 2009

[21] Rhea, Sean: *OpenDHT: A Publicly Accessible DHT Service*. 2009. – URL `http://opendht.org/`. – [Online; Stand 4. Juli 2009]

[22] Schneier, Bruce: *Angewandte Kryptographie*. PEARSON Studium, 2006

[23] Tanenbaum, Andrew S.: *Computer networks*. Prentice Hall, 2002

[24] Wikipedia: *ARP-Spoofing — Wikipedia, Die freie Enzyklopädie*. 2008. – URL `http://de.wikipedia.org/w/index.php?title=ARP-Spoofing&oldid=53197649`. – [Online; Stand 17. Februar 2009]

[25] Wikipedia: *Electronic Code Book Mode — Wikipedia, Die freie Enzyklopädie*. 2008. – URL `http://de.wikipedia.org/w/index.php?title=Electronic_Code_Book_Mode&oldid=53682701`. – [Online; Stand 28. Januar 2009]

[26] Wikipedia: *Bencode — Wikipedia, The Free Encyclopedia*. 2009. – URL `http://en.wikipedia.org/w/index.php?title=Bencode&oldid=297596549`. – [Online; Stand 7. Juli 2009]