NAT Traversal und verlässliche Datenverteilung in geschichteten Peer-to-Peer Systemen

Von der Fakultät für Ingenieurwissenschaften,
Abteilung Informatik und Angewandte Kognitionswissenschaft
der Universität Duisburg-Essen
zur Erlangung des akademischen Grades

Doktors der Ingenieurwissenschaften

genehmigte Dissertation

von

Sebastian Holzapfel
aus
Wesel

1. Gutachter: Prof. Dr.-Ing. Torben Weis
2. Gutachter: Prof. Dr. Kurt Geihs
Tag der mündlichen Prüfung: 21.11.2012

Inhaltsverzeichnis

Abbildungsverzeichnis

Auflistungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die wachsende Verbreitung von Breitband-Internetzugängen und die enorme Steigerung der Rechenleistung von Computern haben in den letzten Jahren dazu geführt, dass Peer-to-Peer-basierte Anwendungen zunehmend an Bedeutung gewonnen haben. Insbesondere im Bereich des Dateiaustausches und der Internettelefonie sind Peer-to-Peer-basierte Anwendungen wie BitTorrent oder Skype weit verbreitet. Im Gegensatz zu klassischen Client-Server Systemen existieren in Peer-to-Peer Systemen keine oder nur sehr wenige Server, so dass alle Aufgaben der Server von den Rechnern der teilnehmenden Nutzer (Peers) übernommen werden müssen. Durch den Einsatz eines Peer-to-Peer Systems lassen sich so die Anschaffungs- und Betriebskosten der Server, die in einem Client-Server System benötigt würden, reduzieren.

Allerdings ergibt sich bei der Entwicklung von Peer-to-Peer-basierten Anwendungen eine Reihe von Herausforderungen. Beispielsweise ist die Entwicklung einer Peer-to-Peer-basierten Anwendung komplexer, da das zugrundeliegende Peer-to-Peer Netzwerk völlig dezentral ist und alle Aufgaben auf die Peers verteilt werden müssen. Zusätzlich kommt bei Peer-to-Peer Systemen erschwerend hinzu, dass Peers zu jedem Zeitpunkt dem System beitreten und es aktiv oder unangekündigt wieder verlassen können. Hierdurch sind zum einen die verfügbaren Ressourcen des Systems nicht konstant und zum anderen erhöht dies wiederum die Komplexität der eingesetzten Algorithmen. Des Weiteren können Peers aufgrund verschiedener Computer und Internetzugänge eine hohe Heterogenität bezüglich der Bandbreite, Rechenkraft, etc. aufweisen.

Im Rahmen des peers@play Projekts [3, 86] wird eine virtuelle Welt entwickelt, die vollständig verteilt auf Basis einer Peer-to-Peer Systemarchitektur arbeitet. Die Entwicklung einer solchen virtuellen Welt umfasst hierbei verschiedene Bereiche. Die vorliegende Arbeit betrachtet hierbei drei Aspekte aus unterschiedlichen Forschungsgebieten.

Der erste Aspekt, der im Rahmen dieser Arbeit betrachtet wird, betrifft die Kommunikation zwischen den einzelnen Peers mit sogenannten NAT Traversal Verfahren. Anschließend wird eine Schichtenarchitektur betrachtet, die eine Peer-to-Peer Anwendung in verschiedene Schichten und Komponenten aufteilt und die Entwicklung und Simulation dieser vereinfacht. Abschließend werden ein Peer-to-Peer Netzwerk sowie ein Replikationsverfahren vorgestellt, die es erlauben Daten einer virtuellen Welt verlässlich zu verteilen.

1.1 NAT Traversal und verlässliche Datenverteilung in geschichteten Peer-to-Peer Systemen

Der erste Aspekt, der im Rahmen dieser Arbeit betrachtet wird, betrifft die Kommunikation zwischen Peers. Diese Kommunikation wird durch die Verbreitung von *Network Address Translation* (NAT) Routern erschwert. NAT Router reduzieren die Konnektivität der Rechner, die sich hinter den NAT Routern befinden. Die Herausforderung besteht darin, den Nachrichtenaustausch zwischen Peers, die sich hinter NAT Routern befinden, zu gewährleisten. Verfahren, die dies ermöglichen, werden als *NAT Traversal* Verfahren bezeichnet. In dieser Arbeit werden ein neues NAT Traversal Verfahren sowie ein Protokoll, das es ermöglicht das NAT Traversal Verfahren auszuwählen, das in einer bestimmten Situation die höchste Erfolgswahrscheinlichkeit bietet, vorgestellt.

Eine weitere Herausforderung betrifft die Umsetzung und konkrete Entwicklung einer Peer-to-Peer-basierten Anwendung. Hier existiert oft eine Vielzahl unterschiedlicher Ansätze für unterschiedliche Problemstellungen. Dabei ist es nicht immer im Vorfeld ersichtlich, welche Kombination von diesen Ansätzen die Beste in Bezug auf Performanz oder Stabilität ist. Um dies zu evaluieren, werden häufig Ansätze zuerst für existierende Simulationsumgebungen implementiert und anschließend für die eigentliche Anwendung. Im Rahmen dieser Arbeit wird eine geschichtete Softwarearchitektur vorgestellt, die eine Peer-to-Peer-basierte Anwendung in Schichten und Komponenten unterteilt. Hierdurch können verschiedene Ansätze einfach ausgetauscht werden. Zusätzlich ermöglicht es die Architektur denselben Quelltext für die eigentliche Anwendung als auch für Simulationen zu verwenden.

Der dritte Aspekt dieser Arbeit betrifft die Datenspeicherung. In einem Serverbasierten System wird der Zustand einer virtuellen Welt auf einem zentralen Server

gespeichert. In einem Peer-to-Peer System hingegen muss der Zustand der virtuellen Welt auf den Peers des Peer-to-Peer Netzwerks gespeichert werden. Hierbei müssen eine Reihe von Anforderungen bezüglich der Verlässlichkeit und Sicherheit der gespeicherten Daten erfüllt werden. Zusätzlich müssen Daten effizient abgefragt werden können. Daher werden im letzten Teil dieser Arbeit ein Peer-to-Peer Netzwerk sowie ein Replikationsverfahren vorgestellt, die die Grundlage für einen sicheren und verlässlichen Peer-to-Peer-basierten Datenspeicher bilden.

1.1.1 NAT Traversal

Die meisten Computer sind heutzutage mittels sogenannter NAT Router mit dem Internet verbunden. Diese NAT Router erschweren es die Rechner, die sich hinter ihnen befinden, direkt zu kontaktieren. Das liegt dran, dass aus dem Internet nur der NAT Router sichtbar ist, aber nicht die dahinter liegenden Rechner. In einer klassischen Client-Server Anwendung ist dies kein Problem, da der Client, der sich hinter einem NAT Router befindet, einen Server im öffentlichen Netzwerk kontaktiert. Im Gegensatz dazu müssen in einer Peer-to-Peer-basierten Anwendung allerdings alle Nachrichten direkt zwischen den Peers ausgetauscht werden. Daher ist es in Peer-to-Peer-basierten Anwendungen notwendig, dass Peers, die sich hinter einem NAT Router befinden, direkt kontaktiert werden können.

Zu diesem Zweck wurden verschiedene NAT Traversal Verfahren [11, 27] entwickelt, die es erlauben einen Peer hinter einem NAT Router direkt zu kontaktieren. Diese Verfahren nutzen unterschiedliche Eigenschaften der involvierten NAT Router aus um einen Nachrichtenaustausch zu ermöglichen. Allerdings verhalten sich NAT Router unterschiedlich. Daher ist die Wahrscheinlichkeit, dass ein Nachrichtenaustausch zwischen zwei Peers ermöglicht werden kann, stark abhängig vom verwendeten NAT Traversal Verfahren und den involvierten NAT Routern. Die Herausforderung besteht darin trotz unterschiedlicher Verhaltensweisen der involvierten Router einen Nachrichtenaustausch zu ermöglichen.

In dieser Arbeit wird ein neues NAT Traversal Verfahren vorgestellt, das mit einer Vielzahl an NAT Routern funktioniert. Zusätzlich wird ein Protokoll vorgestellt, das bestimmte Eigenschaften eines NAT Routers ermittelt. Basierend auf diesen Eigenschaften wird aus einer Menge an verfügbaren NAT Traversal Verfahren das Verfahren ausgewählt, welches die höchste Erfolgswahrscheinlichkeit bietet.

1.1.2 Architektur für komplexe Peer-to-Peer Systeme

Die Entwicklung des peers@play Projektes zeigte, dass die Entwicklung einer komplexen Peer-to-Peer Anwendung in Form einer virtuellen Welt mehr erfordert als das Erstellen einer Netzwerkstruktur und eines Routing Algorithmus. Ein weit verbreiteter Ansatz hierfür im Bereich der verteilten Systeme besteht darin die Anwendung in verschiedene Schichten oder Komponenten aufzuteilen [5, 22]. Diese Aufteilung muss allerdings einige Anforderungen erfüllen.

Im Bereich der Peer-to-Peer Netzwerke existieren verschiedene Ansätze, die sich in verschiedenen Merkmalen unterscheiden. Bei diesen Merkmalen handelt es sich beispielsweise um statische oder dynamische Positionen eines Peers im Peer-to-Peer Netzwerk oder um die Anzahl der Dimensionen des verwendeten Adressraums. Im peers@play Projekt werden verschiedene Peer-to-Peer Netzwerke für verschiedene Aufgaben verwendet. Die Komponenten und Schichten der Architektur müssen daher so entworfen sein, dass mehrere zeitgleich aktive Peer-to-Peer Netzwerk unterstützt werden.

Durch eine geeignete Aufteilung lassen sich unterschiedliche Lösungen für bestimmte Probleme austauschen. Um zu ermitteln, wie sich ein bestimmter Ansatz bei einer steigenden Anzahl an Peers verhält und welche Auswirkung dieser Ansatz auf die Gesamtleistung hat, werden diese Ansätze im Allgemeinen simuliert. Daher werden zwei verschiedene Implementierungen desselben Konzeptes benötigt: eine für die Simulation und eine für die eigentliche Anwendung.

Um dieses Problem zu lösen wird in dieser Arbeit eine Schichtenarchitektur vorgestellt, die es ermöglicht denselben Quelltext für Simulationen und Messungen sowie für die Produktivanwendung zu nutzen. Zusätzlich unterstützt die vorgestellte Schichtenarchitektur mehrere zeitgleich aktive Peer-to-Peer Netzwerke.

1.1.3 Verlässliche Datenverteilung

Eine Herausforderung bei der Entwicklung einer Peer-to-Peer-basierten virtuellen Welt ist das sichere und verlässliche Speichern von Daten. Der Grund hierfür ist, dass bei Peer-to-Peer-basierten Datenspeichern die Daten verteilt auf den Peers gespeichert werden. Allerdings sind Peers nicht zuverlässig, da sie zu jeder Zeit das Netzwerk unangekündigt verlassen können und es kann nicht sichergestellt werden, dass Peers sich immer ehrlich verhalten. Daher kann nicht angenommen werden, dass Daten, die auf einem Peer gespeichert wurden, unverändert und verfügbar bleiben.

Eine virtuelle Welt und das Verhalten in dieser unterliegt gewissen Spielregeln, die unter anderem festlegen welche Aktionen unter welche Bedingungen durchgeführt werden dürfen. Diese Regeln gelten für alle Nutzer der virtuellen Welt und kein Nutzer darf in der Lage sein die Spielregeln zu seinen Gunsten zu umgehen. Eine Möglichkeit die Spielregeln zu umgehen besteht beispielsweise darin den aktuellen Zustand der virtuellen Welt zu fälschen.

Daher muss die Sicherheit und Verfügbarkeit der gespeicherten Daten gewährleistet werden. Zusätzlich zur Sicherheit und Verlässlichkeit der Daten muss es eine Möglichkeit geben, die gespeicherten Daten effizient abzufragen. Peers in einer virtuellen Welt sind in der Regel an Daten interessiert, die sich in einem bestimmten Gebiet befinden. Um die Daten einer Region oder eines Gebietes effizient abzufragen, müssen die Daten mit Positionsbezug gespeichert werden.

Im Rahmen dieser Arbeit wird ein Peer-to-Peer Netzwerk vorgestellt, das auf Voronoi-Diagrammen (Zerlegung eines n-dimensionalen Raumes in Polytope) basiert. Hierdurch können Positions-basierte Daten effizient verteilt und abgefragt werden. Um die Verfügbarkeit der Daten zu gewährleisten werden diese durch ein spezielles Replikationsverfahren mehrfach auf verschiedenen Peers gespeichert. Durch ein spezielles Routingverfahren kann jedes dieser Replikate auf einem anderen Routingpfad erreicht werden, wodurch die Verfügbarkeit der Daten erhöht wird.

1.2 Wissenschaftlicher Beitrag

In dieser Arbeit wird eine Reihe von Ansätzen behandelt, die für die Entwicklung einer Peer-to-Peer-basierten virtuellen Welt relevant sind. Die wissenschaftlichen Beträge dieser Arbeit werden im Folgenden skizziert.

NAT Traversal

- Analyse verschiedener existierender NAT Traversal Mechanismen.
- Entwicklung eines NAT Traversal Verfahrens für das Transmission Control Protocol (TCP) auf Basis von selbst erzeugten TCP Paketen.
- Entwicklung eines Protokolls zur Ermittlung bestimmter Eigenschaften von NAT Routern (MFB Protokoll).
- Adaption des MFB Protokolls zum Einsatz in vollständig dezentralen Systemen.

- Entwicklung eines Regelwerks zur Auswahl des NAT Traversal Verfahrens mit der höchsten Erfolgswahrscheinlichkeit in Abhängigkeit der Eigenschaften der involvierten NAT Router.
- Entwicklung einer Testumgebung für NAT Traversal Verfahren.

Architektur für komplexe Peer-to-Peer Systeme

- Entwicklung einer Softwarearchitektur für komplexe Peer-to-Peer Systeme, die es unter Anderem erlaubt dieselbe Softwarebasis für Simulationszwecke und für den Produktiveinsatz zu verwenden. Weitere Eigenschaften der Architektur sind die Wiederverwendbarkeit und Austauschbarkeit von Komponenten sowie die Unterstützung für mehrere zeitgleich aktive Peer-to-Peer Netzwerke.
- Entwicklung eines Kommunikationsframeworks für Peer-to-Peer Systeme (Peer-to-Peer Link Management).

Verlässliche Datenverteilung

- Symmetrisches Replikationsverfahren für Voronoi-basierte Peer-to-Peer Netzwerke.
- Entwicklung eines Präfix-basierten Routingverfahrens für Voronoi-basierte Peer-to-Peer Netzwerke, das es ermöglicht bestimmte Positionen über disjunkte Routingpfade zu erreichen.

1.3 Organisation der Arbeit

Der Fokus dieser Arbeit liegt auf NAT Traversal Verfahren und der verlässlichen Datenverteilung in Peer-to-Peer Systemen. Die Arbeit unterteilt sich hierbei in sechs Kapitel, die im Folgenden kurz beschrieben werden.

Kapitel 2: Grundlagen

Im Anschluss an die Einleitung werden die Grundlagen, die für das Verständnis der Arbeit notwendig sind, beschrieben. Hierbei werden Netzwerkprotokolle, die Adressumsetzung in Netzwerken und daraus resultierende Schwierigkeiten und Lösungen dargestellt. Im Anschluss wird das Peer-to-Peer System Pastry sowie das in Pastry

verwendete Präfix-Routing beschrieben. Des Weiteren werden Voronoi-Diagramme, die Delaunay-Triangulation sowie Algorithmen um diese zu berechnen beschrieben. Abschließend wird in diesem Kapitel Gears4Net, ein asynchrones und nicht blockierendes Programmiermodell zur Entwicklung von skalierbaren und verteilten Applikationen, vorgestellt.

Kapitel 3: NAT Traversal

Im dritten Kapitel wird ein NAT Traversal Verfahren namens SYNI vorgestellt, das auf der Injektion eines TCP SYN Pakets basiert. Anschließend werden verschiedene existierende TCP NAT Traversal Verfahren analysiert und ihre Anforderungen für eine erfolgreiche Durchführung zusammengefasst. Basierend auf diesen Anforderungen wird ein Protokoll vorgestellt, welches die relevanten Eigenschaften von NAT Routern ermittelt. Unter Verwendung dieser Informationen sowie der Anforderungen der einzelnen TCP NAT Traversal Verfahren kann das TCP NAT Traversal Verfahren mit der höchsten Erfolgswahrscheinlichkeit ausgewählt werden. Das daraus resultierende Regelwerk wird in Abschnitt 3.3.4 vorgestellt. Anschließend wird eine Testumgebung vorgestellt mit der die in diesem Kapitel vorgestellten Verfahren evaluiert wurden. In Abschnitt 3.5 werden die in diesem Kapitel vorgestellten Verfahren evaluiert und abschließend werden die Ergebnisse dieses Kapitels zusammengefasst.

Kapitel 4: Architektur für komplexe Peer-to-Peer Systeme

Im vierten Kapitel wird eine Softwarearchitektur zur Entwicklung und Simulation komplexer Peer-to-Peer Systeme vorgestellt. Diese Softwarearchitektur zerlegt Peer-to-Peer-basierte Anwendungen in verschiedene Schichten und Komponenten. In Abschnitt 4.2 werden die einzelnen Schichten und Komponenten betrachtet. Hierbei werden insbesondere die Komponente für das Verbindungsmanagement sowie die Komponenten für die Simulation näher beschrieben. Abschließend wird mit Hilfe einer Evaluation gezeigt, dass die vorgestellte Softwarearchitektur effizient umgesetzt werden kann.

Kapitel 5: Verlässliche Datenverteilung

Das fünfte Kapitel beschreibt ein Peer-to-Peer Netzwerk basierend auf Voronoi-Diagrammen sowie ein symmetrisches Replikationsverfahren. Das Peer-to-Peer Netzwerk verwendet ein Präfix-basiertes Routing-Verfahren, das es ermöglicht über disjunkte Routingpfade zu verschiedenen Positionen im Netzwerk zu routen. Anschließend wird ein symmetrisches Replikationsverfahren betrachtet, das Daten so repliziert, dass diese mit Hilfe des Routing-Verfahrens über disjunkte Routingpfade erreicht werden können. Abschließend wird in der Evaluation gezeigt, welchen Einfluss böswillige Peers auf die Verfügbarkeit der Daten haben.

Kapitel 6: Zusammenfassung und weiterer Forschungsbedarf

Im letzten Kapitel dieser Arbeit werden die wichtigsten Ergebnisse zusammengefasst und ein Ausblick auf weitere Arbeiten in diesem Gebiet gegeben.

Kapitel 2

Grundlagen

Im diesem Kapitel werden die grundlegenden Protokolle und Verfahren, die für das Verständnis der vorliegenden Arbeit notwendig sind, näher beschrieben. Dabei werden zunächst Protokolle aus der Internet- und der Transportschicht des hybriden Referenzmodells von Tanenbaum [81] vorgestellt. Darauf aufbauend wird die Adressumsetzung in Netzwerken [78], die daraus resultierenden Probleme in Verbindung mit dem Nachrichtenaustausch sowie existierende Lösungen näher beschrieben. Abschließend werden unter anderem das Peer-to-Peer System Pastry und die Grundlagen von Voronoi-Diagrammen näher betrachtet.

2.1 Netzwerkprotokolle

Im Verlauf dieser Arbeit werden einige Details bestimmter Netzwerkprotokolle verwendet, die im folgenden Abschnitt näher beschrieben werden. Die beschriebenen Protokolle gehören jeweils zu unterschiedlichen Schichten des hybriden Referenzmodells von Tanenbaum [81] (siehe Abbildung 2.1). Hierbei wird das der Internetschicht zugehörige Internet Protocol (IP) [62] und die darauf aufbauenden Protokolle TCP (Transmission Control Protocol) [63], UDP (User Datagram Protocol) [60] und ICMP (Internet Control Message Protocol) [61] betrachtet. Diese Protokolle stellen jeweils unterschiedliche Dienste zur Verfügung und verwenden jeweils ein eigenes Datenformat, das in den jeweiligen Protokoll Headern spezifiziert wird.

2.1.1 TCP/IP Modell

Das hybride Referenzmodell von Tanenbaum [81] ist eine Kombination des ISO/OSI Schichtenmodells [22] und des TCP/IP Referenzmodells [14] und besteht aus fünf Schichten (siehe Abbildung 2.1).

Abbildung 2.1: Hybrides Referenzmodell nach Tanenbaum [81]

Jede Schicht dient einer anderen Aufgabe und innerhalb einer jeden Schicht gibt es unterschiedliche Protokolle. Die Transportschicht dient zur Kommunikation zwischen verschiedenen Anwendungen, wohingegen die Internetschicht für die Kommunikation zwischen verschiedenen nicht direkt verbundenen Endgeräten verantwortlich ist. Die Netzwerkschicht dient der Kommunikation zwischen direkt verbundenen Endgeräten und die physikalische Schicht umfasst die Aufgabe Daten über verschiedene Übertragungsmedien (z.B. Kabel oder Funk) zu senden. Im Folgenden werden die wichtigsten Protokolle näher betrachtet.

2.1.2 Internet Protocol

Das Internet Protocol (IP) [62] bildet die Grundlage zur Kommunikation im Internet und ist Teil der Internetschicht. IP existiert in zwei verschiedenen Versionen *IP Version 4 (IPv4)* und *IP Version 6 (IPv6)*. Im Rahmen dieser Arbeit wird ausschließlich IPv4 verwendet, so dass im Folgenden die Bezeichnung IP gleichbedeutend mit IPv4 verwendet wird.

Allgemein betrachtet wird IP verwendet, um Nachrichten zwischen nicht direkt verbundenen Rechnern auszutauschen. Nachrichten werden hierbei von einem Quellrechner über mehrere Zwischenstationen, die sich in unterschiedlichen physikalischen Netzwerken befinden können, zum Zielrechner weitergeleitet. Hierfür erhält jeder Rechner eine eindeutige 32-Bit große IP Adresse, die in vier Blöcke zu je 8 Bit

unterteilt ist. Jede Nachricht, die mittels IP gesendet wird, enthält einen 20 Byte großen Header (siehe Abbildung 2.2).

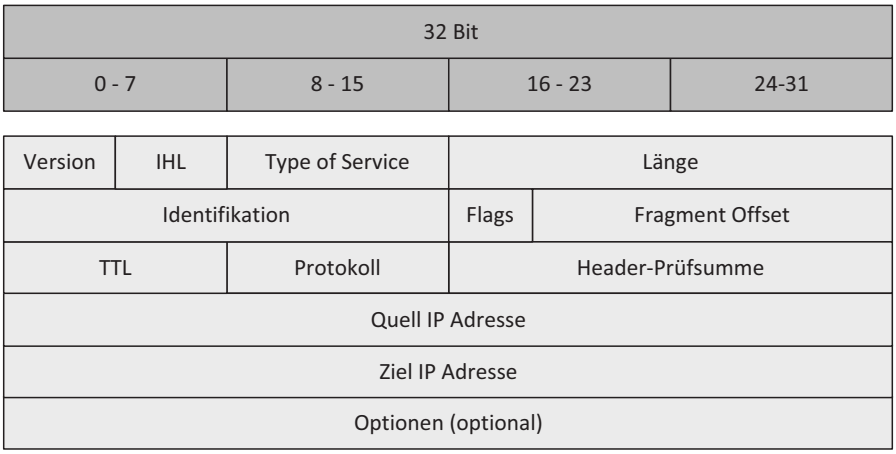


Abbildung 2.2: IP Header Informationen [62]

Der Header enthält eine Reihe von Feldern, wobei im Folgenden nur die Felder näher betrachtet werden, die für diese Arbeit relevant sind. Grundlegend für den Nachrichtenaustausch sind die Felder Ziel- und Quell IP Adresse, die den Absender sowie den Zielrechner angeben. Aufgrund dessen, dass IP Pakete in der Regel über mehrere Stationen bis zum Zielrechner weitergeleitet werden, muss verhindert werden, dass unzustellbare Nachrichten für immer im Netzwerk verbleiben bzw. kreisen. Hierfür wird das Feld Time-to-Live (TTL) verwendet. Dieses Feld wird von jeder weiterleitenden Station dekrementiert. Erreicht die TTL den Wert 0 bevor das Paket das Ziel erreicht hat, wird die Nachricht verworfen und eine ICMP Nachricht (siehe Abschnitt 2.1.5) an den Absender geschickt, die signalisiert, dass die Nachricht verworfen wurde.

2.1.3 User Datagram Protocol

Das User Datagram Protocol (UDP) [60] ist ein einfaches und verbindungsloses Netzwerkprotokoll, das auf IP aufbaut. Im Gegensatz zu IP, das Nachrichten zwischen Rechnern austauscht, wird UDP verwendet, um Nachrichten zwischen verschiedenen Prozessen auf verschiedenen Rechnern auszutauschen. Zu diesem Zweck wird bei UDP zusätzlich zu der IP Adresse eine Portnummer verwendet, die den Prozess auf einem Rechner adressiert. Aufgrund dessen, dass UDP ein verbindungsloses Protokoll ist, wird keine explizite Verbindung aufgebaut bevor Nachtrichten an einen entfernten Prozess gesendet werden. Die Nachrichten (Datagramme) werden direkt

an eine Ziel IP Adresse und einen Zielport gesendet. Die Kombination aus IP Adresse und Portnummer wird im Folgenden als IP Endpunkt bezeichnet. Der Absender einer Nachricht erhält bei UDP keinerlei Bestätigung darüber, ob die Daten beim Empfänger angekommen sind oder nicht. Des Weiteren ist es bei UDP möglich, dass sich Nachrichten auf dem Weg zum Ziel überholen und so in einer anderen Reihenfolge ankommen als sie gesendet wurden. Darüber hinaus ist es möglich, dass der Sender Nachrichten schneller sendet, als der Empfänger diese verarbeiten kann, da UDP keine Mechanismen für eine Flusskontrolle bietet. In diesem Fall können Nachrichten verloren gehen.

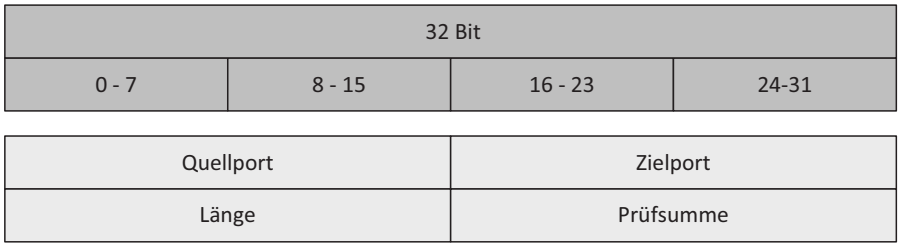


Abbildung 2.3: UDP Header Informationen [60]

Aufgrund des geringen Funktionsumfangs des Protokolls, besteht der Nachrichten Header im Wesentlichen aus dem Quell- und Zielport sowie der Gesamtgröße des Datagramms (siehe Abbildung 2.3).

2.1.4 Transmission Control Protocol

Das Transmission Control Protocol (TCP) [63] basiert wie UDP auf IP, ist aber ein verbindungsorientiertes und zuverlässiges Transportprotokoll. Im Vergleich zu UDP verwendet TCP einen größeren Header (siehe Abbildung 2.4) mit mehr Informationen sowie eine Menge von Algorithmen zur Steuerung des Nachrichtenflusses.

TCP adressiert ebenso wie UDP Prozesse auf entfernten Rechnern garantiert dabei aber, dass jedes Paket in der richtigen Reihenfolge und fehlerfrei beim Empfänger ankommt. Um dies zu gewährleisten, verwendet TCP unter anderem Sequenznummern und Bestätigungsnummern (siehe Abbildung 2.4), so dass der Empfänger die erhaltenen Daten sortieren und quittieren kann. In welchen Abständen diese Quittierung erfolgt, ist abhängig von der eingesetzten Flusskontrolle (Fenstermechanismus). Hierbei wird ein Fenster definiert, das angibt, wie viele Nachrichten gesendet werden dürfen bis eine Quittierung erfolgen muss. Damit zwei Kommunikationspartner Nachrichten zueinander senden können, müssen sie im Gegensatz zu UDP erst

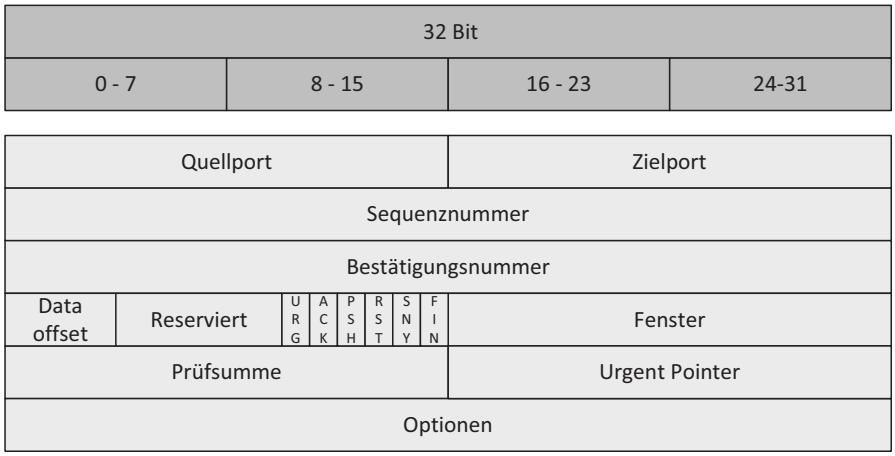


Abbildung 2.4: TCP Header Informationen [63]

eine Verbindung zueinander aufbauen. Dieser Verbindungsaufbau erfolgt in einem 3-Wege-Handshake, wie in Abbildung 2.5 zu sehen ist.

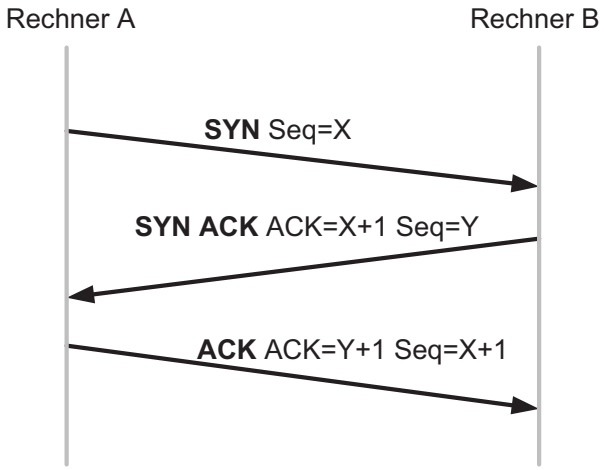


Abbildung 2.5: TCP Handshake beim Verbindungsaufbau [63]

Dabei sendet Rechner *A*, der eine Verbindung zum Zielrechner *B* aufbauen möchte, ein Paket an *B*. Dieses Paket beinhaltet unter anderem die Quell- und Zieladresse aus dem IP Header und den Quell- und Zielport aus dem TCP Header. Des Weiteren ist das SYN Flag im TCP Header und eine initiale Sequenznummer (ISN) *X* gesetzt (siehe Abbildung 2.5). *B* antwortet daraufhin mit einem Paket in dem das SYN und das ACK Flag gesetzt sind. Zusätzlich zu der von *B* gewählten ISN *Y* bestätigt *B* mit der Bestätigungsnummer *X+1*, dass er das vorherige Paket erhalten hat. Um den Verbindungsaufbau abzuschließen, sendet *A* ein Paket mit gesetztem ACK Flag und bestätigt mit der Bestätigungsnummer *Y+1* das vorherige Paket von *B*. Nachdem

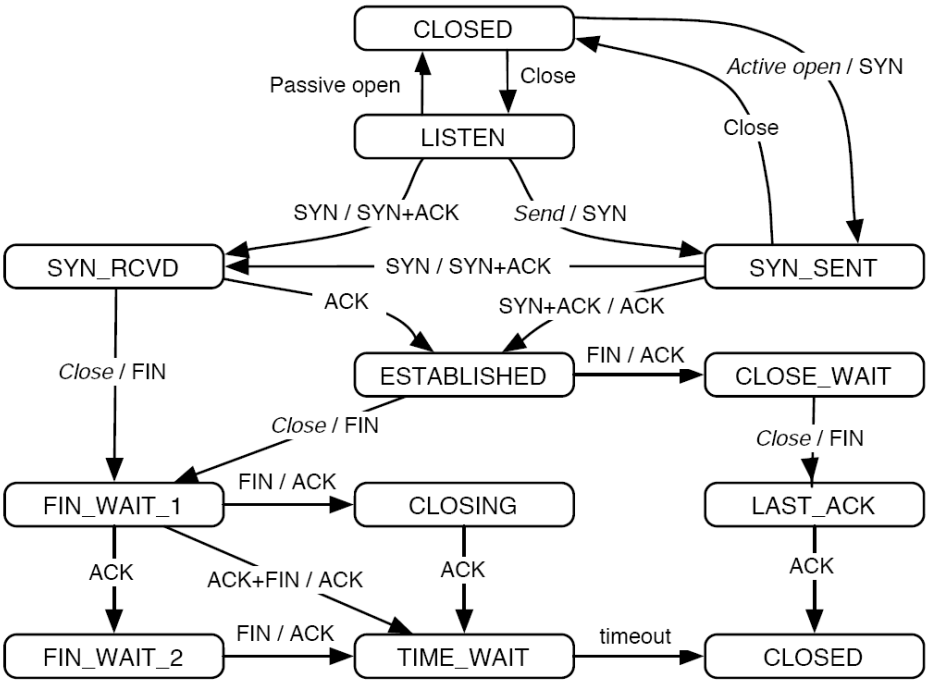


Abbildung 2.6: TCP Zustandsdiagramm [1, 63]

die Verbindung aufgebaut wurde, können über diese Verbindung Daten gesendet werden.

Eine Besonderheit beim Verbindungsaufbau ist das simultane Öffnen einer Verbindung. Hierbei initiieren A und B jeweils einen Verbindungsaufbau, so dass beide Rechner ein TCP Paket mit gesetztem SYN Flag senden. A und B befinden sich in dem Zustand *SYN SENT* (siehe Abbildung 2.6) und warten auf einen entsprechendes TCP Paket mit gesetztem SYN und ACK Flag. Allerdings empfangen beide ein TCP Paket bei dem nur das SYN Flag gesetzt ist. An dieser Stelle wird erkannt, dass es sich um einen simultanen Verbindungsaufbau handelt und der Zustand wird entsprechend auf *SYN RECEIVED* gewechselt. Anschließend wird der Verbindungsaufbau, wie in Abbildung 2.6 dargestellt, fortgesetzt.

Um eine Verbindung aktiv zu schließen, wird vergleichbar mit dem Verbindungsaufbau ein Handshake durchgeführt. Allerdings wird im Gegensatz zum Verbindungsaufbau in diesem Fall ein 4-Wege-Handshake durchgeführt und anstatt des SYN Flags wird das FIN Flag gesetzt. Möchte A die Verbindung schließen, sendet A ein TCP Paket mit gesetztem FIN Flag. B antwortet darauf mit einem entsprechenden ACK und kann anschließend noch weitere Daten senden. Sobald B keine Daten mehr senden möchte, sendet B ebenfalls ein FIN und A wird mit einem ACK ant-

worten. Anschließend ist die Verbindung geschlossen. Nachdem die letzte Nachricht für den Verbindungsabbau ausgetauscht wurde, befindet sich der Kommunikationspartner, der den Verbindungsabbau initiiert hat, im *TIME_WAIT* Zustand (siehe Abbildung 2.6). In diesem Zustand verbleibt der Kommunikationspartner abhängig vom Betriebssystem für einige Sekunden oder Minuten. Solange der Kommunikationspartner in diesem Zustand ist, kann er keine neue Verbindung mit den gleichen IP Endpunkten initiieren, da diese noch in den Verbindungstabellen des Betriebssystems stehen.

Allerdings ist es möglich, dass aufgrund von technischen Problemen eine Verbindung nicht mehr aktiv geschlossen werden kann, weil beispielsweise die Bestätigungsnachrichten verloren gehen. In diesem Fall kann die Verbindung abgebrochen werden. Hierfür wird eine Nachricht mit gesetztem Reset Flag gesendet. Wird eine Verbindung mittels Reset Flag beendet, kann sofort wieder versucht werden die Verbindung mit den gleichen IP Endpunkten aufzubauen. Das Reset-Flag kann auch zur Abweisung unerwünschter Verbindungen verwendet werden oder wenn beispielsweise eine Nachricht an eine Portnummer gesendet wurde, auf der nicht auf eingehende Nachrichten gewartet wird.

2.1.5 Internet Control Message Protocol

Ein weiteres Protokoll, das auf IP aufbaut, ist das Internet Control Message Protocol (ICMP) [61]. Dieses Protokoll ist im Gegensatz zu TCP und UDP kein Transportprotokoll im eigentlichen Sinne, sondern ein Kontrollprotokoll zum Austausch von Diagnoseinformationen und Fehlermeldungen.

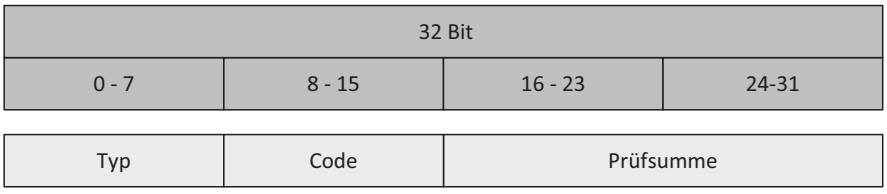


Abbildung 2.7: ICMP Header Informationen [61]

ICMP hat wie UDP einen sehr kleinen Nachrichten Header, der in Abbildung 2.7 dargestellt ist. Die Nachrichten lassen sich in verschiedene Kategorien (Typen) einteilen. Zusätzlich zum Typ der Nachricht, enthält jede Nachricht noch einen Code, der die Art der Nachricht genauer spezifiziert.

Wenn beispielsweise eine Nachricht an eine Adresse gesendet wird, auf der kein Rechner auf Nachrichten wartet, wird eine ICMP Nachricht vom Typ *Ziel nicht erreichbar* mit dem Code *Port nicht erreichbar* gesendet. Im Fall, dass eine Nachricht nicht zugestellt werden konnte, bevor die TTL abgelaufen ist, wird eine Nachricht vom Typ *Zeitlimit überschritten* mit dem Code *Lebensdauer abgelaufen* gesendet. In [61] ist vollständige Auflistung aller Nachrichten Typen und Codes gegeben.

2.2 Firewall

Eine Firewall ist eine Sicherheitskomponente, deren Aufgabe darin besteht ein internes Netzwerk nach außen hin zu schützen. Um beispielsweise ein privates Netzwerk gegenüber dem Internet zu schützen, muss die Firewall genau zwischen beiden Netzwerken positioniert werden. Eine Firewall kann aus verschiedenen Komponenten bestehen. Im Folgenden werden die Komponenten näher betrachtet, die für diese Arbeit relevant sind.

2.2.1 Paketfilter

Ein Paketfilter ist eine Komponente einer Firewall. Hierbei wird der gesamte ein- und ausgehende Netzwerkverkehr auf Paketebene gefiltert und anhand eines Regelwerks, das beispielsweise von einem Netzwerkadministrator erstellt wird, entschieden, wie auf die einzelnen Paketen reagiert wird. Der Paketfilter wertet zu diesem Zweck Informationen aus der Internet- sowie aus der Transportschicht aus. Zusätzlich werden auch Informationen von ICMP Nachrichten ausgewertet.

Eine Regel eines Paketfilters kann die folgenden Informationen auswerten und angeben welche Aktion (erlauben oder ablehnen) ausgeführt werden soll.

- Quell- und Zieladresse (IP)
- Protokolltyp (TCP, UDP oder ICMP)
- Quell- und Zielport (TCP und UDP)
- Flags aus dem Header (TCP)
- Nachrichtentyp (ICMP)
- Netzwerkinterface

Wenn eine Anfrage aus dem internen Netzwerk gesendet wird, wird hierfür automatisch eine neue Regel hinzugefügt, so dass Antworten auf diese Anfrage vom Paketfilter weitergeleitet werden. Dazu werden Verbindungstabellen geführt, in denen gespeichert wird, welcher interne Rechner mit welchem Externen kommuniziert. Dabei werden neben den IP Adressen auch die verwendeten Portnummern und der Protokolltyp gespeichert.

2.2.2 Zustandsgesteuerte Paketfilterung

Eine weitere Form des Paketfilters ist die sogenannte zustandsgesteuerte Paketfilterung. Hierbei wird zu den bereits genannten Informationen der Zustand einer Verbindung überprüft. Wenn beispielsweise ein TCP Paket mit gesetztem SYN Flag gesendet wurde, muss darauf ein Paket mit gesetztem SYN und ACK Flag folgen. Ohne eine genaue Überprüfung des Zustands würde der Paketfilter auch ein Paket ins das interne Netzwerk weiterleiten, das nur das SYN oder nur das ACK Flag gesetzt hat. Die zustandsgesteuerte Paketfilterung erlaubt es noch genauer den Datenfluss zu kontrollieren.

2.2.3 IP-Spoofing

Mit dem Begriff IP-Spoofing bezeichnet man in Computernetzen das Versenden von IP Paketen mit gefälschter Absender IP Adresse. Um IP-Spoofing zu unterbinden bzw. zu erschweren können beispielsweise zwischen zwei verschiedenen Netzwerken A und B Ingress [26] und Egress Filter eingesetzt werden. Ingress Filter blockieren Pakete, die aus einem Netzwerk B kommen, aber eine Absender Adresse aus Netzwerk A enthalten. Egress Filter hingegen blockieren Pakete, die aus dem Netzwerk A kommen, aber keine gültige Absender Adresse aus dem Netzwerk A haben.

TCP verwendet, wie in Abschnitt 2.1.4 beschrieben, eine Sequenznummer für jedes Paket. Hierdurch wird ebenfalls IP-Spoofing erschwert, weil nur Pakete mit gültiger Sequenznummer akzeptiert werden. Um zu verhindern, dass die vom Betriebssystem verwendete Implementierung zur Bestimmung der initialen Sequenznummer zu einfach erraten werden kann, bieten einige Firewalls die Möglichkeit einen zufälligen Offset auf diese Sequenznummer zu addieren oder zu subtrahieren. In diesem Fall muss die Firewall die Sequenznummer von jedem ein- und ausgehenden Paket ändern.

2.3 Adressumsetzung in Netzwerken

Unter einer Adressumsetzung in Netzwerken (engl. Network Address Translation, NAT) [25] versteht man die Übersetzung von IP Adressen eines Adressraumes in einen Anderen. Daher wird NAT häufig beim Übergang von zwei verschiedenen Computernetzwerken eingesetzt. Dabei wird bei jedem Datenpaket die IP Adresse des anfragenden Computers durch die IP Adresse des NAT Gerätes ersetzt. Die Antworten erhält wiederum das NAT Gerät und muss diese dann an den entsprechenden Computer weiterleiten.

Die Adressumsetzung in Netzwerken ist nicht standardisiert, daher gibt es eine Reihe verschiedener Verfahren. Allerdings sollten alle diese Verfahren die folgenden Eigenschaften [78] besitzen:

- Transparente Adresszuordnung
- Transparentes Weiterleiten von Paketen durch die Adressumsetzung
- Zuordnung von ICMP Nachrichten

Die einfachste Variante eines NAT Verfahrens ist die, in der nur die IP Adressen ersetzt werden. Hierbei wird allerdings für jeden Rechner, der eine Verbindung zum externen Netzwerk aufbauen möchte, eine externe IP Adresse benötigt.

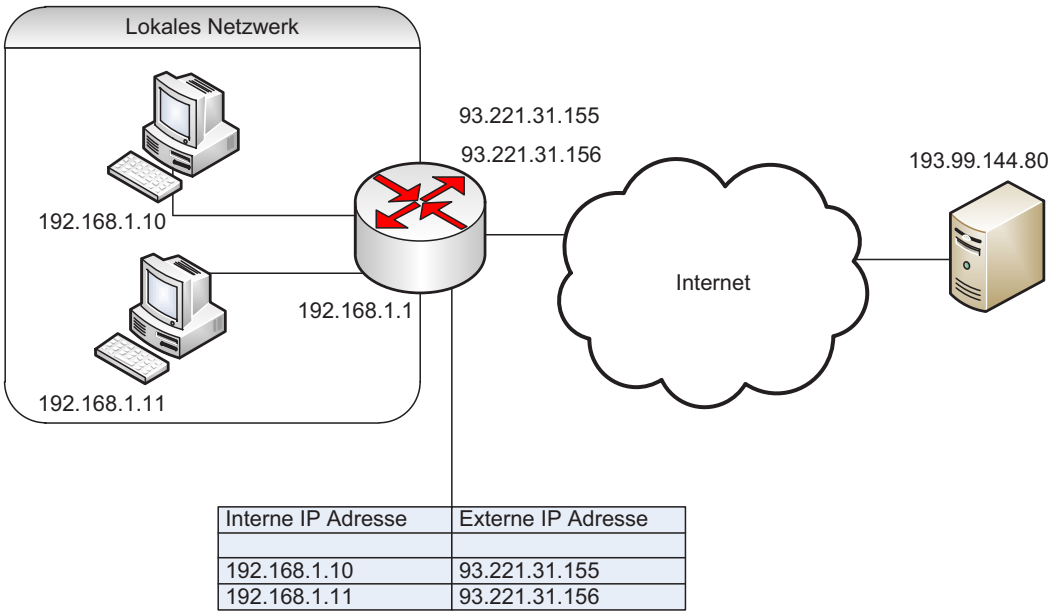


Abbildung 2.8: Adressumsetzung in Netzwerken

Dieses Verfahren ist in Abbildung 2.8 grafisch dargestellt. In diesem Szenario wollen die beiden Rechner mit den internen IP Adressen 192.168.1.10 und 192.168.1.11 Daten an einen Rechner im externen Netzwerk mit der IP Adresse 193.99.144.80 senden. Der von den beiden Rechnern verwendete NAT Router verfügt über zwei externe IP Adressen. Wenn die Rechner das erste Mal Daten senden, erstellt der NAT Router eine Zuordnung zwischen der internen IP Adresse des anfragenden Rechners und einer seiner externen IP Adressen. In dem betrachteten Beispiel wird beispielsweise dem Rechner mit der internen IP Adresse 192.168.1.10 die externe IP Adresse 93.221.31.155 zugewiesen und dem Rechner mit der internen IP Adresse 192.168.1.11 wird die externe IP Adresse 93.221.31.156 zugewiesen. Hierdurch sind beide Rechner in der Lage mit Rechnern aus dem externen Netzwerk zu kommunizieren. Wäre noch ein dritter Rechner im internen Netzwerk, könnten trotzdem immer nur zwei Rechner gleichzeitig mit dem externen Netzwerk kommunizieren, da eine externe IP Adresse immer exklusiv an einen internen Rechner gebunden wird.

2.4 Adress- und Portumsetzung in Netzwerken

Eine Variante des im vorherigen Abschnitt vorgestellten NAT Verfahrens ist die Adress- und Portumsetzung in Netzwerken. Diese unterscheidet sich vom einfachen NAT in dem Punkt, dass nicht nur die IP Adresse geändert wird, sondern zusätzlich noch die Portnummer. Im weiteren Verlauf dieser Arbeit wird der Begriff NAT für die Adress- und Portumsetzung in Netzwerken verwendet.

Dieses Verfahren hat den Vorteil, dass sich mehrere interne Rechner eine externe IP Adresse teilen können. Zusätzlich erlaubt es einen sparsameren Umgang mit öffentlichen IP Adressen und die Maskierung des dahinter befindlichen Netzwerkes. Durch den Einsatz von NAT sind die Computer, die sich hinter einem NAT Router befinden, nicht mehr direkt erreichbar sind (siehe nachfolgenden Abschnitt). Dies kann man als Vor- aber auch als Nachteil von NAT ansehen.

In den folgenden Abschnitten werden verschiedene Verhaltensweisen und Eigenschaften [7, 31, 32] von NAT Routern beschrieben. Da das Verhalten von NAT Routern nicht standardisiert ist, kann es von NAT Router zu NAT Router verschieden sein. Allerdings eignen sich die verwendeten Verhaltensweisen und Eigenschaften um das Verhalten der meisten NAT Router zu beschreiben.

2.4.1 Portzuordnung

Im Folgenden wird für das Tupel bestehend aus einer IP Adresse und einer Portnummer der Begriff *IP Endpunkt* verwendet. Es wird davon ausgegangen, dass sich ein Computer *A* hinter einem NAT Router befindet. *A* befindet sich entsprechend in einem privaten Netzwerk mit einem privaten Adressraum und möchte Pakete an ein öffentliches Netzwerk senden. Dabei verwendet *A* für jedes ausgehende Paket den gleichen lokalen IP Endpunkt, um Pakete an verschiedene Computer im öffentlichen Netzwerk zu senden. Die Portzuordnung beschreibt hierbei die Übersetzung des lokalen IP Endpunkts von A auf den externen IP Endpunkt des NAT Routers und umgekehrt. In den meisten Fällen verfügt ein NAT Router nur über eine externe IP Adresse *E*, so dass die externe IP Adresse für alle erstellten Portzuordnungen gleich ist. Daher kann der NAT Router nur entscheiden, welche externe Portnummer dem lokalen IP Endpunkt zugeordnet werden soll.

Zuordnungsstrategie	(Quelle, Ziel) → Extern
Endpunkt-unabhängig	*(A:a, beliebig:beliebig) → E:e*
Adressen-abhängig	*(A:a, X:beliebig) → E:e*
Port-abhängig	*(A:a, beliebig:x) → E:e*
Adressen- und Port-abhängig	*(A:a, X:x) → E:e*

Tabelle 2.1: Portzuordnung für ausgehende Pakete

Bei der Zuordnung der externen Portnummern können verschiedene Strategien angewendet werden. Eine mögliche Strategie ist es beispielsweise die gleiche externe Portnummer zu verwenden, solange der lokale IP Endpunkt konstant bleibt. Der IP Endpunkt des Ziels wird hierbei nicht ausgewertet, daher spricht man bei dieser Strategie von einer *Endpunkt-unabhängigen Zuordnung*. Im Gegensatz zu dieser Strategie kann die Ziel IP Adresse *X*, die Zielportnummer *x* oder der Ziel IP Endpunkt *X:x* ausgewertet werden, um zu entscheiden welche externe Portnummer verwendet wird. Verwendet der NAT Router eine *Adressen- und Port-abhängige Zuordnung* wird für jeden neuen Ziel IP Endpunkt eine neue Portnummer verwendet. Hierbei kann der NAT Router eine zufällige Portnummer für jede Zuordnung auswählen oder versuchen die lokale Portnummer zu erhalten, wenn diese noch nicht zugeordnet wurde. In dem Fall, dass die ausgewählte externe Portnummer schon an einen lokalen IP Endpunkt gebunden ist, kann eine zufällige Portnummer oder die nächst freie externe Portnummer (beispielsweise +1 oder -1) ausgewählt werden.

In Tabelle 2.1 sind alle Strategien für die Portzuordnung dargestellt. Diese Strategien gelten für die Transportprotokolle TCP und UDP.

2.4.2 Filterung

Während die Portzuordnung das Verhalten des NAT Routers bei der Zuordnung neuer Portnummern für ausgehende Pakete beschreibt, beschreibt die Filterung das Verhalten für eingehende Pakete. Um eingehende Pakete an einen internen Rechner weiterleiten zu können, muss die externe Portnummer, an der das eingehende Paket ankommt, einem internen IP Endpunkt zugeordnet sein. In dem Fall, dass der externen Portnummer kein interner IP Endpunkt zugeordnet ist, wird dieses Paket ausgefiltert, da es nicht weitergeleitet werden kann.

Wenn der externen Portnummer ein interner IP Endpunkt zugeordnet ist, kann der NAT Router den Paketfluss filtern. Werden beispielsweise alle Pakete unabhängig vom Absender an den internen IP Endpunkt weitergeleitet werden, so verwendet der NAT Router eine sogenannte *Endpunkt-unabhängige Filterung*. Überprüft der NAT Router den IP Endpunkt des Absenders, sprich es werden nur Pakete weitergeleitet, die von diesem IP Endpunkt gesendet wurden, so verwendet der NAT Router eine *Adressen- und Port-abhängige Filterung*.

In Tabelle 2.2 sind die verschiedenen Verhaltensweisen der Filterung von eingehenden Paketen zusammengefasst.

Filterung	Extern ← Ziel
Endpunkt-unabhängig	*E:e* ← *beliebig:beliebig*
Adressen-abhängig	*E:e* ← *X:beliebig*
Port-abhängig	*E:e* ← *beliebig:x*
Adressen- und Port-abhängig	*E:e* ← *X:x*

Tabelle 2.2: NAT Filterung für eingehende Pakete

In den Fällen, in denen der NAT Router eingehende Pakete aufgrund der Filterregeln verwirft, kann er den Absender darüber informieren oder das Paket stillschweigend verwerfen. Wenn der NAT Router den Absender darüber informiert, dass das Paket nicht zugestellt wurde, kann er eine ICMP Fehlernachricht an den Absender schicken. Handelt es sich bei dem eingehenden Paket um ein TCP Paket kann der NAT Router auch eine TCP Nachricht mit gesetztem Reset Flag (RST) an den Absender zurück schicken (siehe Abschnitt 2.1.4).

2.5 Simple Traversal of UDP Through NAT

Das *Simple Traversal of UDP Through NAT* Protokoll (STUN), das im folgenden Abschnitt näher betrachtet wird, wurde erstmals in [69] spezifiziert. In einer nachfolgenden Spezifikation [68] wurden allerdings einige Protokolldetails grundlegend geändert, so dass sich die Versionen des STUN Protokolls in [69] und in [68] deutlich unterscheiden. Im Rahmen dieser Arbeit werden auf einige der in der ursprünglichen Spezifikation verwendeten Informationen zurückgegriffen, die in der nachfolgenden Spezifikation [68] nicht mehr enthalten sind. Damit deutlich wird, welche Version des STUN Protokolls im weiteren Verlauf der Arbeit gemeint ist, wird vom klassischen STUN Protokoll gesprochen, wenn das STUN Protokoll in der ursprünglichen Spezifikation [69] gemeint ist.

Das klassische STUN Protokoll ist ein simples auf UDP basierendes Anfrage/Antwort Protokoll, das es einem Rechner (klassischer STUN Client) ermöglicht eine Reihe von Informationen über seine Netzwerkumgebung zu ermitteln. Der klassische STUN Client sendet hierfür eine Reihe von Anfragen an einen öffentlich erreichbaren klassischen STUN Server. Anhand der erhaltenen Antworten, die der klassische STUN Client empfängt, kann er ermitteln, ob er sich hinter einem NAT Gerät befindet und wie sich das NAT Gerät verhält. Um das Verhalten eines NAT Gerätes zu beschreiben verwendet das klassische STUN Protokoll eine eigene Klassifikation, die in Abschnitt 2.5.1 beschrieben wird. Insgesamt ermöglicht das klassische STUN Protokoll einem Rechner die folgenden Informationen über seine Netzwerkumgebung zu ermitteln:

- Externe IP Adresse
- Externe Portnummer, die zur Kommunikation mit dem STUN Server verwendet wurde
- NAT Klassifikation

Bevor der genaue Ablauf des Protokolls in Abschnitt 2.5.3 beschrieben wird, wird im Folgenden die Klassifikation der NAT Typen sowie die Spezifikation der Nachrichtenformate, wie sie im klassischen STUN-Protokoll definiert wurden, näher beschrieben.

2.5.1 NAT Klassifikation

Das klassische STUN Protokoll unterscheidet beim Verhalten von NAT Geräten vier verschiedene Typen. Jeder dieser Typen lässt sich auch als Kombination aus dem Portzuordnungs- und dem Filterverhalten, wie sie in Abschnitt 2.4 beschrieben wurden, beschreiben. Die einzelnen NAT Typen des klassischen STUN Protokolls bauen aufeinander auf und werden in jeder Stufe restriktiver.

Klassischer STUN Typ	Portzuordnung	Filterung
Full Cone	Endpunkt-unabhängig	Endpunkt-unabhängig
Restricted Cone	Endpunkt-unabhängig	Adressen-abhängig
Port Restricted Cone	Endpunkt-unabhängig	Adressen- und Port-abhängig
Symmetric	Adressen- und Port-abhängig	Adressen- und Port-abhängig

Tabelle 2.3: Klassische STUN Klassifikation und das zugehörige Portzuordnungs- und Filterverhalten

In Tabelle 2.3 sind die 4 NAT Typen und die entsprechenden Portzuordnungs- und Filterverhalten dargestellt. Alle Cone Varianten beschreiben beispielsweise ein Endpunkt-unabhängiges Zuordnungsverhalten und ein jeweils anderes Filterverhalten. Das Symmetric NAT hingegen ist äquivalent zu einem Adressen- und Port-abhängigen Portzuordnungs- und Filterverhalten.

Aufgrund der Tatsache, dass viele NAT Geräte nicht exakt mit diesen Typen klassifiziert werden können, wurde diese Klassifikation im STUN Protokoll vollständig verworfen [68].

2.5.2 Protokoll Spezifikation

Dieser Abschnitt beschreibt die grundlegende Spezifikation der verwendeten Nachrichten im klassischen STUN Protokoll. Bei den klassischen STUN Nachrichten wird zwischen *Binding* und *Shared Secret* Nachrichten unterschieden (siehe Tabelle 2.4). Die *Shared Secret* Nachrichten dienen zur Authentizität und Integrität, haben aber keinen Einfluss auf die Ermittlung der Netzwerkumgebung. Diese Nachrichten können nur über eine TLS Verbindung (Transport Layer Security) [23] gesendet werden und werden nicht von allen klassischen STUN Servern unterstützt. Daher werden im Folgenden nur die *Binding* Nachrichten näher betrachtet.

Klassische STUN Nachrichten besitzen einen 20 Byte großen Header. Dieser Header teilt sich auf in zwei Byte für den Nachrichtentyp, zwei Byte für die Nachrichtenlänge und 16 Byte für einen Transaktionsidentifikator (TID). Nach dem Header folgt eine Menge von klassischen STUN Nachrichtenattributen (siehe Tabelle 2.5), die je nach Nachrichtentyp gesetzt werden. Im Folgenden werden die Attribute, die im Laufe der Arbeit verwendet werden, näher erklärt. Eine vollständige Beschreibung aller Attribute und wann diese verwendet werden, ist in [69] gegeben.

0x0001	Binding Request	0x0002	Shared Secret Request
0x0101	Binding Response	0x0102	Shared Secret Response
0x0111	Binding Error Response	0x0112	Shared Secret Error Response

Tabelle 2.4: Klassische STUN Nachrichtentypen

Das Attribut *MAPPED-ADDRESS* wird vom Server in einer Binding Response Nachricht gesetzt und beinhaltet die Adresse, von der der Server den Binding Request empfangen hat.

0x0001	MAPPED-ADDRESS	0x0006	USERNAME
0x0002	RESPONSE-ADDRESS	0x0007	PASSWORD
0x0003	CHANGE-REQUEST	0x0008	MESSAGE-INTEGRITY
0x0004	SOURCE-ADDRESS	0x0009	ERROR-CODE
0x0005	CHANGED-ADDRESS	0x000a	UNKNOWN-ATTRIBUTES
		0x000b	REFLECTED-FROM

Tabelle 2.5: Klassische STUN Nachrichtenattribute

Eine Besonderheit eines klassischen STUN Servers ist, dass er über zwei öffentliche IP Adressen verfügt. Dadurch kann der Server über verschiedene IP Endpunkte Nachrichten an einen Client senden. Durch das *CHANGE-REQUEST* Attribut kann einen klassischer STUN Client den Server anweisen von welchem IP Endpunkt aus er die Antwort senden soll. Wenn im *CHANGE-REQUEST* Attribut nichts angegeben ist, sendet der Server eine Antwort über den ersten IP Endpunkt. Durch das Setzen zweier Flags im Attribut wird der Server aufgefordert über eine andere IP Adresse (ChangeIP), einen anderen Port (ChangePort) oder eine andere IP Adresse und einen anderen Port (ChangeIP & ChangePort) zu antworten.

Im *CHANGED-ADDRESS* Attribut ist die Adresse des Servers angegeben, die bei gesetztem ChangeIP und ChangePort Flag benutzt wird.

2.5.3 Ermittlung der Umgebung

Dieser Abschnitt beschreibt den genauen Ablauf des klassischen STUN Protokolls. Im vorherigen Abschnitt 2.5.1 wurden die einzelnen NAT Typen eingeführt und näher beschrieben. Zusätzlich zu diesen NAT Typen unterscheidet das klassische STUN Protokoll auch die im Einsatz befindliche Firewall. Im Folgenden sind alle möglichen Netzwerkumgebungen, die das klassische STUN Protokoll ermitteln kann, aufgelistet.

- Offenes Internet
- Firewall, die keinen ausgehenden UDP Verkehr gestattet
- Symmetric UDP Firewall (erlaubt ausgehenden UDP Verkehr mit Adressen- und Port-abhängigem Filterverhalten)
- Full Cone NAT
- Restricted Cone NAT
- Port Restricted Cone NAT
- Symmetric NAT

Um die Netzwerkumgebung zu ermitteln, führt der Client eine Reihe von Tests aus. Der genaue Ablauf ist in Abbildung 2.9 schematisch dargestellt. Die einzelnen Tests, die in der Abbildung dargestellt sind, werden im folgenden Abschnitt beschrieben.

Tests

Um die Netzwerkumgebung, in der sich ein klassischer STUN Client befindet, ermitteln zu können, sendet der Client eine Reihe von *STUN Binding Requests* mit verschiedenen Parametern (siehe Abschnitt 2.5). Eine spezielle Anfrage wird im Folgenden als Test bezeichnet. Aufgrund der Tatsache, dass alle Tests bzw. Anfragen mittels UDP durchgeführt werden, sollten diese bei ausbleibender Antwort wiederholt werden, um so eine Verfälschung des Ergebnisses durch verloren gegangene Pakete zu vermeiden.

Test 1 Beim ersten Test wird ein *STUN Binding Request* an einen klassischen STUN Server gesendet, der weder ein Flag beim Attribut *CHANGE-REQUEST*, noch das Attribut *RESPONSE-ADDRESS* gesetzt hat. Der Server sendet daraufhin einen *STUN Binding Response* an den IP Endpunkt zurück, von dem er den *STUN Binding Request* erhalten hat.

Test 2 Der zweite Test sendet einen *STUN Binding Request*, bei dem die beiden Flags ChangeIP und ChangePort des Attributs *CHANGE-REQUEST* gesetzt sind. Der Server sendet daraufhin einen *STUN Binding Response* über seine zweite IP Adresse und seinen zweiten Port an den Client.

Test 3 Hier wird der klassische STUN Server aufgefordert einen *STUN Binding Response* über einen anderen Port, als an den gesendet wurde, zu antworten. Dafür sendet der Client einen *STUN Binding Request*, bei dem nur das Flag ChangePort des Attributs *CHANGE-REQUEST* gesetzt ist.

Protokollablauf

Die Ermittlung der Umgebung beginnt damit, dass der Client den ersten Test ausführt. Erhält der Client auf diese Anfrage keine Antwort, blockiert die *Firewall ausgehenden UDP Verkehr* und das Protokoll terminiert. Wenn der Client hingegen einen *STUN Binding Response* erhält, vergleicht er seinen internen IP Endpunkt mit dem IP Endpunkt aus dem *MAPPED-ADDRESS* Attribut. Entspricht der interne IP Endpunkt dem aus dem Attribut, befindet sich der Client nicht hinter einem NAT Gerät. In beiden Fällen wird mit Test 2 fortgefahren. Zuerst wird der Fall betrachtet, in dem beide IP Endpunkte identisch sind.

Wird vom 2. Test keine Antwort erhalten, befindet sich der Client hinter einer Symmetric UDP Firewall. Erhält der Client hingegen eine Antwort auf die Anfrage des 2. Tests, ist zwischen dem Client und dem Internet keine Firewall und kein NAT Gerät. Der Client ist also direkt im Internet (Offenes Internet). In beiden Fällen terminiert das Protokoll nach diesem Test.

Waren die beiden IP Endpunkte vom 1. Test nicht identisch, weiß der Client, dass er sich auf jeden Fall hinter einem NAT Gerät befindet und es werden mit dem 2. Test andere Ergebnisse ermittelt. Für den Fall, dass der Client eine Antwort erhalten hat, befindet er sich hinter einem *Full Cone* NAT, weil er von zwei verschiedenen IP Endpunkten Pakete empfangen konnte und er nur zum ersten IP Endpunkt selber ein Paket gesendet hat. In dem Fall, dass der Client keine Antwort erhalten hat, führt er den 1. Test noch einmal durch, sendet aber diese Anfrage an die andere Adresse des klassischen STUN Servers, die im Attribut *CHANGED-ADDRESS* der Antwort des ersten Tests angegeben war.

Nachdem der Client eine Antwort auf diesen Test erhalten hat, wird der IP Endpunkt aus dem *MAPPED-ADDRESS* Attribut mit dem vom 1. Test verglichen. Sind

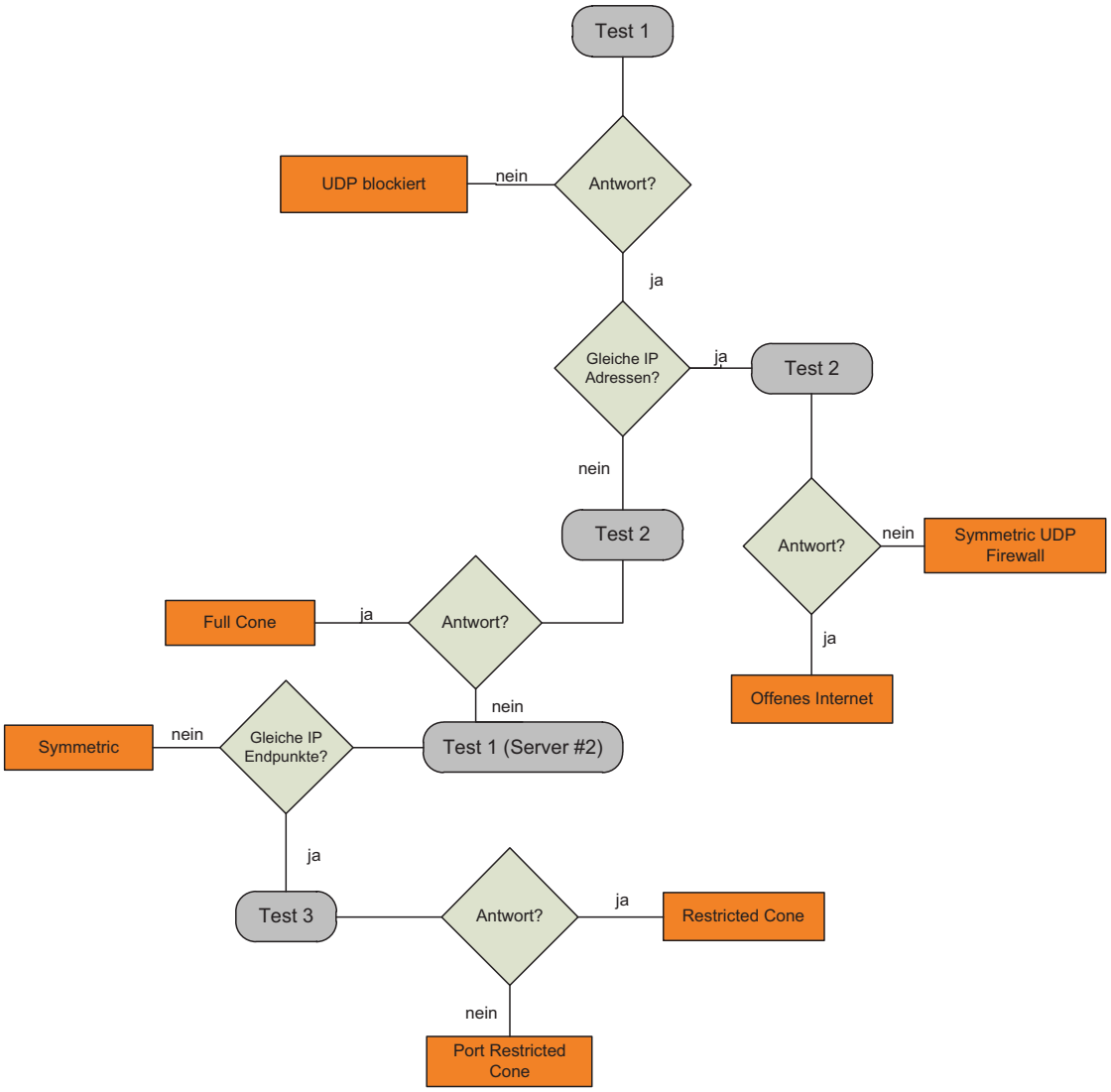


Abbildung 2.9: Ablaufplan bei der Ermittlung der Netzumgebung [69]

die beiden IP Endpunkte unterschiedlich, befindet sich der Client hinter einem Symmetric NAT, weil zwei verschiedene externe IP Endpunkte für den gleichen internen IP Endpunkt verwendet wurden.

Sollten die beiden IP Endpunkte übereinstimmen, so befindet sich der Client entweder hinter einem *Restricted Cone* oder *Port Restricted Cone* NAT. Um zu entscheiden welcher NAT Typ es ist, führt der Client den 3. Test aus. Erhält der Client auf den 3. Test eine Antwort, so befindet er sich hinter einem *Restricted Cone*, ansonsten hinter einem *Port Restricted Cone* NAT.

2.6 NAT Traversal

In diesem Abschnitt werden die Schwierigkeiten beim Verbindungsaufbau zwischen zwei Rechnern in Verbindung mit NAT Routern sowie entsprechende Lösungsmöglichkeiten vorgestellt. Im Folgenden wird davon ausgegangen, dass zwei Rechner A und B Pakete miteinander austauschen möchten. Zu diesem Zweck müssen A und B auf einem bestimmten lokalen IP Endpunkt LE_A und LE_B auf eingehende Pakete warten. Wenn sich A und B beispielsweise im gleichen lokalen Netzwerk befinden, können sie direkt Pakete an die jeweiligen IP Endpunkte schicken. Befinden sich A und B aber hinter den NAT Routern R_A und R_B ist dies so nicht mehr möglich. Wenn A ein Paket an B schicken möchte, muss das Paket an R_B adressiert werden. Zu diesem Zweck muss A die IP Adresse von R_B kennen sowie die entsprechende externe Portnummer. Diese externe Portnummer muss dem internen IP Endpunkt LE_B zugeordnet sein, damit R_B das Paket weiterleiten kann. Des Weiteren muss eine entsprechende Filterregel existieren, so dass R_B die Pakete an B weiterleiten kann.

Eine Möglichkeit eine entsprechenden Portzuordung und Filterregel zu erstellen ist das manuelle Erstellen durch den Benutzer. Allerdings ist das manuelle Erstellen eine relativ hohe Hürde für den Benutzer, so dass eine Methode ohne manuelles Eingreifen zu bevorzugen ist. Die im Folgenden betrachteten NAT Traversal Mechanismen benötigen kein manuelles Eingreifen des Benutzers.

2.6.1 Universal Plug and Play

Universal Plug and Play (UPnP) [84] dient zur Ansteuerung von verschieden Geräten (Router, Drucker, ...) von verschiedenen Herstellern über ein IP-basiertes Netzwerk. Das *Internet Gateway Device* (IGD) Protokoll, welches Teil des UPnP Protokolls ist, ermöglicht es eine Portzuordnung und dazugehörige Filterregel programmatisch zu erstellen. UPnP ist die effektivste und einfachste Möglichkeit verglichen mit anderen NAT Traversal Mechanismen. Allerdings ist UPnP nicht bei allen NAT Routern verfügbar, sei es durch mangelnde Unterstützung oder wegen Sicherheitsangelegenheiten.

Um zu überprüfen, ob sich ein UPnP fähiger NAT Router im Netzwerk befindet, sendet der Rechner im lokalen Netzwerk eine Nachricht per UDP an die Multicast Adresse *239.255.255.250:1900*. Diese Nachricht muss die Nutzdaten wie in Auflistung 2.1 enthalten.

```
1    M-SEARCH * HTTP/1.1
2    HOST: 239.255.255.250:1900
3    MAN: "ssdp:discover"
4    MX: seconds to delay response
5    ST: search target
```

Auflistung 2.1: M-Search Anfrage

Wenn sich im lokalen Netzwerk ein NAT Router befindet, der UPnP unterstützt und aktiviert hat, wird er mit einer Nachricht antworten, die die folgenden Nutzdaten (siehe Auflistung 2.2) enthält.

```
1    HTTP/1.1 200 OK
2    CACHE-CONTROL: max-age = seconds until advertisement expires
3    DATE: when response was generated
4    EXT:
5    LOCATION: URL for UPnP description for root device
6    SERVER: OS/version UPnP/1.1 product/version
7    ST: search target
8    USN: composite identifier for the advertisement
9    BOOTID.UPNP.ORG: number increased each time device sends an initial announce
         or an update message
```

Auflistung 2.2: M-Search Antwort

Die wichtigste Information aus dieser Nachricht ist die URL zu dem Feld *Location*. Das unter dieser URL befindliche XML Dokument ist eine Beschreibung des UPnP Gerätes und beinhaltet Informationen über Hersteller, Modell und einen Service Typen, der angibt um was für ein Gerät (z.B. Router) es sich handelt.

```
1    <deviceType>urn:schemas-upnp-org:device:InternetGatewayDevice:1</deviceType>
2    <friendlyName>D-Link DI Series</friendlyName>
3    <manufacturer>D-Link</manufacturer>
4    <manufacturerURL>http://www.dlink.com</manufacturerURL>
5    <modelDescription>D-Link Internet Gateway Device</modelDescription>
6    <modelName>D-Link Internet Gateway Device</modelName>
7    <UDN>uuid:00-0D-88-E7-A8-15-028B14AC0</UDN>
8    <presentationURL>http://172.20.139.2:80/</presentationURL>
9    <serviceList>
10      <service>
11         <serviceType>urn:schemas-upnp-org:service:Layer3Forwarding:1</
               serviceType>
12         <serviceId>urn:upnporg:serviceId:L3Forwarding1</serviceId>
13         <controlURL>/upnp/control1</controlURL>
14         <eventSubURL>/Layer3Forwarding</eventSubURL>
15         <SCPDURL>http://172.20.139.2:80/serv1.xml</SCPDURL>
16      </service>
17   </serviceList>
```

Auflistung 2.3: Auszug aus der XML-Beschreibung eines UPnP fähigen Gerätes

In Auflistung 2.3 ist ein Auszug der XML Beschreibung eines NAT Routers gegeben. In der Beschreibung finden sich weitere Verweise auf XML Dokumente, die die unterstützen Dienste beinhalten (siehe Auflistung 2.4).

```
1    <action>
2       <name>AddPortMapping</name>
3       <argumentList></argumentList>
4    </action>
5    <action>
6       <name>GetExternalIPAddress</name>
7       <argumentList></argumentList>
8    </action>
```

Auflistung 2.4: Auszug aus der XML-Beschreibung der unterstützten Dienste

Damit eine Portzuordnung erstellt werden kann und alle eingehenden Pakete an einen internen Rechner weitergeleitet werden können, muss der NAT Router den Dienst *AddPortMapping* zur Verfügung stellen. Hiermit kann ein Rechner den Router anweisen eine Portzuordnung für ein bestimmtes Protokoll und für eine bestimmte Zeit zu erstellen und die Pakete an ihn zu schicken. Die aktuell verwendete externe IP Adresse des Routers kann ein Rechner mit Hilfe des Dienstes *GetExternalIPAddress* abfragen.

2.6.2 Verbindungsumkehrung

Ein anderes Verfahren ist die Verbindungsumkehrung (engl. Connection Reversal). Bei diesem Verfahren geht man davon aus, dass ein Rechner A eine Verbindung zu einem Rechner B aufbauen möchte. Allerdings ist B nicht direkt erreichbar und kann auch nicht mittels UPnP oder einem anderen Verfahren eine Portzuordnung erstellen. Auf der anderen Seite ist A aber in der Lage eine entsprechende Portzuordnung zu erstellen. Damit nun eine Verbindung zwischen A und B aufgebaut werden kann, muss A seinen Verbindungswunsch an B leiten. Hierfür müssen A und B über einen gemeinsamen Kommunikationskanal (z.B. ein Server mit dem beide verbunden sind) verfügen. A kann dann hierüber seinen Verbindungswunsch an B leiten und da A eine entsprechende Portzuordnung erstellt hat, kann B eine direkte Verbindung zu A aufbauen.

2.6.3 Hole Punching

Ein weiteres Verfahren, um eine Verbindung zwischen zwei Rechnern hinter verschiedenen NAT Geräten herzustellen, ist das sogenannte Hole Punching [27, 32, 33]. Hierbei wird davon ausgegangen, dass keiner der beiden Kommunikationspartner (Rechner A und Rechner B) eine entsprechende Portzuordnung erstellen kann, wie es beispielsweise mit UPnP möglich wäre. Damit mittels Hole Punching eine Verbindung hergestellt werden kann, ist es notwendig, dass beide NAT Geräte eine Endpunktunabhängige Portzuordnung verwenden. Im Folgenden wird das Hole Punching Verfahren am Beispiel von UDP und TCP näher beschrieben.

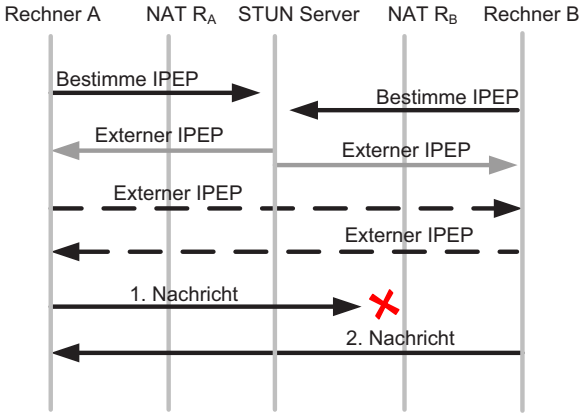


Abbildung 2.10: Hole Punching mittels UDP

UDP Bevor das eigentliche Verfahren durchgeführt werden kann, müssen A und B ihre externen IP Endpunkte (IPEP) ermitteln. Zu diesem Zweck können A und B beispielsweise das klassische STUN Protokoll verwenden. Dabei senden A und B jeweils UDP Pakete an einen öffentlich erreichbaren klassischen STUN Server. Die NAT Router von A und B erstellen dann jeweils eine externe Portzuordnung und entsprechende Filterregeln, so dass der klassische STUN Server Nachrichten an A und B senden kann. Die Antworten, die der klassische STUN Server sendet, beinhalten den externen IP Endpunkt von A bzw. von B.

Nachdem A und B ihren externen IP Endpunkt ermittelt haben, tauschen sie die Information darüber aus. Wenn A den externen IP Endpunkt von B erhalten hat, sendet A ein Paket an den externen IP Endpunkt von B. Hierbei ist es wichtig, dass der NAT Router von A denselben externen IP Endpunkt verwendet, der auch zur Kommunikation mit dem klassischen STUN Server verwendet wurde. R_A erstellt beim Senden dieser Nachricht eine weitere Filterregel, so dass B von seinem externen

IP Endpunkt antworten kann. Das von A gesendete Paket wird allerdings den NAT Router von B nicht passieren, da R_B keine entsprechende Filterregel erstellt hat. Wenn B jetzt allerdings ein Paket von seinem IP Endpunkt an A sendet, erstellt R_B eine Filterregel, so dass A Nachrichten von seinem externen IP Endpunkt an B senden kann. Zusätzlich sollte das Paket von B den NAT Router von A passieren, da dieser bereits eine entsprechende Filterregel erstellt hat. A und B sind ab diesem Zeitpunkt in der Lage Nachrichten miteinander auszutauschen.

TCP Die Grundidee dieses Verfahrens lässt sich auch auf das Transportprotokoll TCP übertragen. Allerdings ist das Verfahren aufwendiger und die Erfolgswahrscheinlichkeit geringer, da TCP verbindungsorientiert ist.

Beim TCP Hole Punching wird ebenso wie bei UDP vorausgesetzt, dass die beiden externen IP Endpunkte von A und B bekannt sind. Die beiden NAT Router müssen ein Endpunkt-unabhängiges Portzuordnungsverhalten aufweisen, da sonst die ermittelten externen IP Endpunkte nicht verwendet werden können.

Wenn nun A eine TCP Verbindung zu B aufbaut, wird gemäß des 3-Wege Handshakes ein TCP Paket mit gesetztem SYN Flag an den externen IP Endpunkt von B gesendet. Zu diesem Zeitpunkt existiert bei R_B keine entsprechende Filterregel, so dass das Paket nicht an B weitergeleitet wird. R_B kann in diesem Fall eine ICMP Nachricht mit dem Typ *Ziel nicht erreichbar* oder ein TCP Paket mit gesetztem Reset (RST) Flag an A senden. Eine TCP RST Nachricht wird hierbei als fataler Fehler behandelt und kann dazu führen, dass R_A die durch das SYN Paket erstellte Portzuordnung entfernt, was den Versuch des Verbindungsaufbaus beenden würde. Um zu verhindern, dass eine solche TCP RST Nachricht gesendet wird, versuchen die meisten TCP Hole Punching Verfahren (siehe Abschnitt 3.2.2) sicherzustellen, dass die SYN Nachricht den eigenen NAT Router passiert, so dass dieser die Portzuordnung erstellt, aber nicht beim Ziel NAT Router ankommt. Zu diesem Zweck kann der Time-to-Live (TTL) Wert (siehe Abschnitt 2.1.2) so gesetzt werden, dass der TTL Wert der Nachricht den Wert Null erreicht, bevor die Nachricht beim Ziel NAT Router ankommt. Der passende TTL Wert ist hierbei abhängig von der Anzahl der Stationen, die zwischen dem Quell NAT Router und dem Ziel NAT Router liegen. Idealerweise wird der Wert so gesetzt, dass das Paket bis eine Station vor dem Ziel NAT Router geleitet wird, da sich auf dem Pfad zum Ziel noch weitere NAT Router befinden könnten und auch diese eine entsprechende Portzuordnung erstellen müssen. Allerdings befindet sich ein Rechner normalerweise nur hinter einem NAT Router, so dass ein TTL Wert von 2 ausreicht, damit das Paket den eigenen NAT Router passiert, aber nicht das Ziel NAT erreicht. In dem Fall, dass sich ein

Rechner hinteren mehreren NAT Routern (Multi-Level NAT) befindet, muss der entsprechende TTL Wert für jedes Ziel individuell bestimmt werden. Hierfür können beispielsweise so lange IP Pakete mit TCP oder UDP Nutzdaten und inkrementiertem TTL Wert an den Ziel Router gesendet werden, bis keine ICMP Nachricht vom Typ *Zeitlimit überschritten* (siehe Abschnitt 2.1.2) mehr empfangen wird. Im Allgemeinen wird eine ICMP Nachricht vom Typ *Zeitlimit überschritten* im Gegensatz zu einem TCP Reset nicht als fataler Fehler behandelt.

NAT Router verwalten zusätzlich zu den Portzuordnungen und den Filterregeln noch den Zustand einer TCP Verbindung. Hierbei kann der Router durch eine detaillierte Kontrolle aller ein- und ausgehenden Nachrichten einer Verbindung den exakten Zustand bestimmen. Hierdurch ist der Router beispielsweise auch in der Lage beim Verbindungsaufbau festzustellen welches Paket als nächstes folgen muss. Nachdem im betrachteten Beispiel A die Verbindung zu B initiiert hat, muss B ebenfalls einen Verbindungsaufbau zu A initiieren. Hierbei wird wiederum ein TCP Paket mit gesetztem SYN Flag an A gesendet. Wenn dieses Paket bei R_A ankommt, kann R_A das Paket verwerfen, da es nicht dem erwarteten Paket (SYN und ACK Flag gesetzt) des 3-Wege-Handshakes entspricht. R_A kann hierauf wiederum mit einem TCP Reset Paket reagieren.

Die Erfolgswahrscheinlichkeit des Verbindungsaufbaus bei TCP ist im Vergleich zu UDP zusätzlich abhängig vom Filterverhalten in Bezug auf den Zustand einer TCP Verbindung.

2.6.4 Kommunikation über einen Mittelsmann

Eine einfache Lösung für das Verbindungsproblem mit NAT Routern ist das Weiterleiten aller Pakete über einen direkt erreichbaren Mittelsmann (engl. Relaying). Hierbei bauen A und B eine Verbindung zu einem direkt erreichbaren dritten Rechner C (z.B. ein Server) auf. Alle Pakete, die A und B miteinander austauschen, werden zuerst an C gesendet und dieser leitet die Pakete dann weiter. Dieses Verfahren funktioniert unabhängig vom Verhalten von R_A und R_B, da diese nur die ausgehende Verbindung zu C erlauben müssen. Allerdings ist dieses Verfahren ineffizienter und langsamer als eine direkte Verbindung, weil erst alle Pakete vom Absender an C und von C zum Ziel geschickt werden.

2.7 Pastry

Nachdem in den vorherigen Abschnitten die Grundlagen im Bereich der Netzwerkprotokolle und NAT Traversal Verfahren beschrieben wurden, werden im folgenden Abschnitt die wesentlichen Eigenschaften und Verfahren des Peer-to-Peer Systems Pastry [71] betrachtet. Pastry kann als Grundlage für Peer-to-Peer Systeme verwendet werden und stellt dafür eine skalierbare, dezentrale Objekt Lokalisierung sowie ein Routing Verfahren zur Verfügung. Jeder Knoten (Peer) in einem Pastry Netzwerk verfügt über einen eindeutigen numerischen Identifikator (KnotenID). Nachrichten verfügen ebenfalls über einen numerischen Identifikator (Zieladresse) und ein Pastry Knoten leitet Nachrichten an den Pastry Knoten weiter, der die numerisch kürzeste Distanz zwischen seiner KnotenID und der Zieladresse hat. In Pastry ist die erwartete Anzahl an Schritten um eine Nachricht ans Ziel zu senden $O(log(N))$ [71], wobei N die Anzahl der Pastry Knoten im Netzwerk ist.

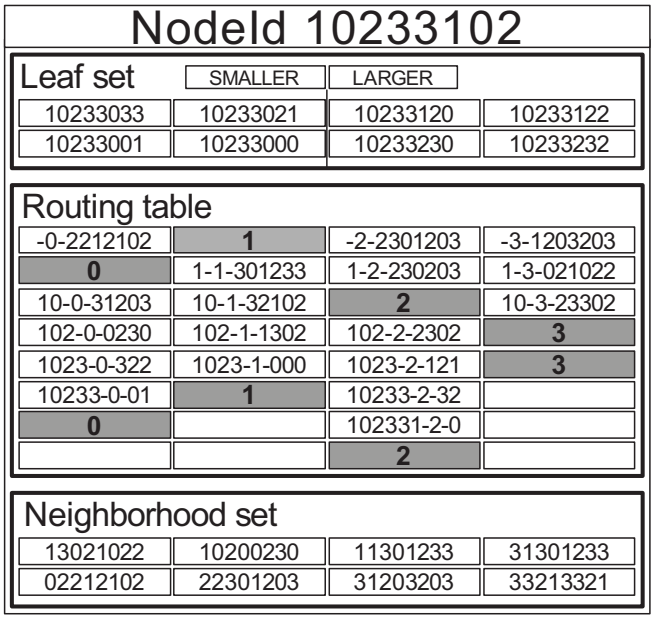
2.7.1 Knoten

Jeder Knoten in einem Pastry Netzwerk verfügt über eine eindeutige 128-Bit große KnotenID. Pastry verwendet hierbei einen 128 Bit großen zirkulären Schlüsselraum, der von 0 bis $2^{128} - 1$ reicht. Diese KnotenIDs geben die Position eines Knotens im Schlüsselraum an und werden zufällig gewählt. Eine Möglichkeit die KnotenIDs zu generieren ist einen Hashwert mit Hilfe der externen IP Adresse des Knotens zu berechnen. In Pastry werden KnotenIDs als Sequenz von Ziffern mit der Basis 2^b dargestellt, wobei b ein Konfigurationsparameter ist (typischer Wert für b ist 4).

Jeder Knoten verwaltet eine Zustandstabelle bestehend aus einer Routing Tabelle, einem Leaf-Set und einer Nachbarschaftsliste. In Abbildung 2.11 ist die Zustandstabelle eines Knotens mit der KnotenID 10233102 dargestellt.

Das Leaf-Set L eines Knotens X enthält hierbei $\frac{|L|}{2}$ Knoten, die numerisch die kürzeste Distanz zum Knoten X haben und größer sind sowie $\frac{|L|}{2}$ Knoten, die numerisch die kürzeste Distanz zum Knoten X haben und kleiner sind. Durch die Verbindungen der Knoten zu den Knoten ihres Leaf-Sets bildet sich eine Ringstruktur welche dafür sorgt, dass jeder Knoten im Pastry Netzwerk erreicht werden kann und sich keine Partitionen bilden.

Die Nachbarschaftstabelle M eines Knotens X enthält im Gegensatz zum Leaf-Set Knoten, die gemäß einer Metrik (z.B. Latenz), die kürzeste Distanz zum Knoten haben. Diese Knoten werden nicht für das Routing verwendet bzw. benötigt, sondern



NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Abbildung 2.11: Zustandstabelle eines Knotens mit der KnotenID 10233102 [71]

können für Dienste verwendet werden, die beispielsweise eine sehr schnelle Antwort benötigen. Typische Werte für $|L|$ und $|M|$ sind 2^b oder $2 \cdot 2^b$ [71].

Die Routing Tabelle eines Pastry Knotens besteht aus $\log_{2^b}(N)$ Zeilen und jede Zeile enthält $2^b - 1$ Einträge. Die $2^b - 1$ Einträge in Zeile n der Routing Tabelle eines Knotens X beinhalten hierbei Knoten, deren ersten n Ziffern gleich mit denen der KnotenID von X sind und an der Stelle $n + 1$ eine andere Ziffer haben. Je höher der gewählte Wert für b ist, desto mehr Einträge enthält die Routing Tabelle. Die Auswahl eines geeigneten Wertes ist hierbei eine Abwägung zwischen der Größe der Routing Tabelle und dem dadurch erhöhten Verwaltungsaufwand und der Anzahl der Schritte, die es braucht bis eine Nachricht sein Ziel erreicht.

2.7.2 Routing

In diesem Abschnitt wird der grundlegende Routingalgorithmus von Pastry (siehe Algorithmus 1) näher betrachtet. Hierbei wird angenommen, dass eine Nachricht mit der Zieladresse ZD bei einem Pastry Knoten mit der KnotenID A ankommt.

Wenn ein Knoten A eine Nachricht empfängt, wird zuerst überprüft, ob die Zieladresse der Nachricht (ZD) innerhalb der Grenzen des eigenen Leaf-Sets liegt. Wenn das der Fall ist, wird der Knoten L_i aus dem Leaf-Set ausgewählt, der die kürzeste

Algorithmus 1 Pseudocode des Routingalgorithmus von Pastry

A : Pastry Knoten mit der KnotenID A

R_l^i : Eintrag der Routing Tabelle in Spalte i und Zeile l

L_i : Eintrag an der Stelle i des Leaf-Sets L

ZD_l : Ziffer der Zieladresse ZD an der Stelle l

$bGP(A, B)$: berechnet die Länge des gemeinsamen Präfixes von A und B

```
if L_{-|L|/2} ≤ ZD ≤ L_{|L|/2} then
  //Zieladresse (ZD) ist im Bereich des Leaf-Sets
  wähle L_i mit |ZD − L_i| ist minimal
  if |ZD − A| > |ZD − L_i| then
    sendeNachricht an L_i
  else
    verarbeiteNachricht
  end if
else
  //Nutzen der Routing Tabelle
  l = bGP(ZD,A)
  if R_l^{ZD_l} ≠ null then
    sendeNachricht an R_l^{ZD_l}
  else
    wähle T ∈ L ∪ R ∪ M mit bGP(T, ZD) ≥ l, |T − ZD| < |A − ZD|
  end if
end if
```

numerische Distanz zu ZD aufweist. Anschließend muss überprüft werden, ob A näher an ZD liegt als der gewählte Knoten L_i. Ist die Distanz von A größer, wird die Nachricht an L_i gesendet und andernfalls wird die Nachricht von A verarbeitet.

Liegt ZD nicht innerhalb des Leaf-Sets wird mit Hilfe der Routing Tabelle entschieden, an welchen Knoten die Nachricht weitergeleitet werden soll. Hierbei muss die Länge des gemeinsamen Präfixes des neuen Knotens und ZD mindestens um eins größer sein als die Länge des gemeinsamen Präfixes von A und ZD.

In einigen Fällen kann es vorkommen, dass der Eintrag $R_l^{ZD_l}$ leer ist. Dann wird die Nachricht an einen Knoten weitergeleitet, dessen gemeinsamer Präfix mit ZD mindestens genauso lang ist wie der von A und ZD, aber numerisch näher an ZD liegt. Solch ein Knoten muss sich im Leaf-Set von A befinden.

Das Routing Verfahren von Pastry konvergiert immer, da in jedem Schritt die Nachricht an einen Knoten weitergeleitet wird, der einen größeren gemeinsamen Präfix mit der Nachricht hat oder numerisch näher an der Zieladresse der Nachricht liegt als der aktuelle Knoten.

2.7.3 Selbstorganisation

In diesem Abschnitt wird beschrieben, wie sich in Pastry neue Knoten in ein bestehendes Pastry Netzwerk integrieren und wie Knoten dieses wieder regelkonform verlassen.

Beitreten

Wenn ein Knoten X einem Pastry Netzwerk beitreten möchte, muss er seine Zustandstabelle füllen und sich im Pastry Netzwerk bekannt machen. Zu diesem Zweck muss X mindestens einen aktiven Knoten A aus dem Pastry Netzwerk kennen. Um einen solchen Knoten kennen zu lernen existieren sogenannte *Bootstrapping-Verfahren* [20, 45, 46]. Nachdem X einen aktiven Knoten A mittel *Bootstrapping-Verfahren* kennengelernt hat, schickt X eine *JOIN*-Nachricht an A. Diese *JOIN*-Nachricht enthält als Zieladresse die KnotenID von X und wird mit Hilfe von A an den Knoten Z geleitet, der numerisch am nächsten an der Adresse X liegt. Die Knoten A, Z sowie alle Knoten auf dem Pfad zwischen A und Z schicken ihre Zustandstabelle an X als Antwort auf dessen *JOIN*-Nachricht.

Nachdem X die Zustandstabellen empfangen hat, kann X seine eigene Zustandstabelle mit Knoten aus den empfangenen Zustandstabellen füllen. Das Leaf-Set von Z dient als Basis für das Leaf-Set von X, da Z numerisch am nächsten an X liegt. Des Weiteren kann X die Nachbarschaftsliste von Z als Grundlage für seine eigene Liste verwenden.

Als nächstes muss X seine Routing Tabelle füllen. Die Einträge für die erste Zeile der Routing Tabelle sind unabhängig von der KnotenID, so dass X die erste Zeile der Routing Tabelle von A übernehmen kann. Die nächste Zeile kann X von der Routing Tabelle von B übernehmen. B ist hierbei der erste Knoten, an den A die *JOIN*-Nachricht von X weitergeleitet hat. X kann diese Zeile übernehmen, da die KnotenIDs von B und X an der ersten Stelle dieselbe Ziffer haben. Die nächsten Zeilen werden dann jeweils von aus den entsprechenden Zeilen der anderen Knoten, die auf dem Pfad von A nach Z lagen, genommen.

Nachdem X seine Zustandstabelle gefüllt hat, sendet X seine Tabelle an alle Knoten, die er in seine Zustandstabelle aufgenommen hat. Die Knoten können dann mit diesen Informationen ihre eigene Zustandstabelle aktualisieren.

Verlassen

Knoten können ausfallen oder das Pastry Netzwerk ohne Warnung verlassen. In diesem Fall müssen alle Knoten, die den ausgefallenen Knoten kannten, ihre Zustandstabelle entsprechend aktualisieren. Pastry überwacht die Knoten aus dem Leaf-Set und der Nachbarschaftsliste periodisch und erkennt Ausfälle in diesen Listen sehr schnell. Ein Ausfall eines Knotens aus der Routing Tabelle wird erst erkannt, wenn der ausgefallene Knoten kontaktiert wird.

Um einen ausgefallenen Knoten aus dem Leaf-Set zu ersetzen, wird der Knoten mit dem höchsten Index auf der richtigen Seite aus dem Leaf-Set nach seinem Leaf-Set gefragt und mit Hilfe dieser Informationen wird der ausgefallene Knoten ersetzt. Dieses Verfahren garantiert, dass jeder Knoten sein Leaf-Set reparieren kann, solange nicht zeitgleich $|L/2|$ Knoten mit angrenzenden KnotenIDs ausfallen. Aufgrund der verwendeten Hashfunktion ist es sehr unwahrscheinlich, dass Knoten mit angrenzenden KnotenIDs in der realen Welt nah beieinander sind. Daher ist die Wahrscheinlichkeit dafür, dass zeitgleich $|L/2|$ unabhängige Knoten ausfallen sehr gering.

Fällt ein Knoten aus der Nachbarschaftsliste aus, werden die anderen Knoten der Nachbarschaftsliste nach ihrer Nachbarschaftsliste gefragt und die eigene Nachbarschaftsliste aktualisiert.

Wenn ein Knoten R_l^d aus der Routing Tabelle ersetzt werden muss, wird eine Nachricht an einen anderen Knoten R_l^i mit $i \neq d$ aus der gleichen Zeile wie der ausgefallene Knoten gesendet und nach dessen Eintrag von R_l^d gefragt. Sollte dieser Knoten keinen entsprechenden Knoten liefern können, wird der nächste Knoten aus der Zeile gefragt. Wenn keiner aus der Zeile einen aktiven Knoten für R_l^d hat wird R_{l+1}^i mit $i \neq d$ gefragt.

2.8 Voronoi-Diagramme

Ein Voronoi-Diagramm [8] beschreibt eine Zerlegung eines n-dimensionalen Raumes in Polytope (Regionen). Diese Zerlegung in die verschiedenen Regionen wird durch

eine diskrete Menge an Punkten (Zentren) bestimmt. Jede Region wird durch genau ein Zentrum beschrieben. Die Region eines Zentrums Z enthält hierbei alle Punkte, die gemäß der euklidischen Metrik näher an Z liegen als an irgendeinem anderem Zentrum. In Abbildung 2.12 ist eine diskrete Menge an Punkten aus einem zweidimensionalen Raum gegeben sowie das daraus resultierende Voronoi-Diagramm dargestellt.

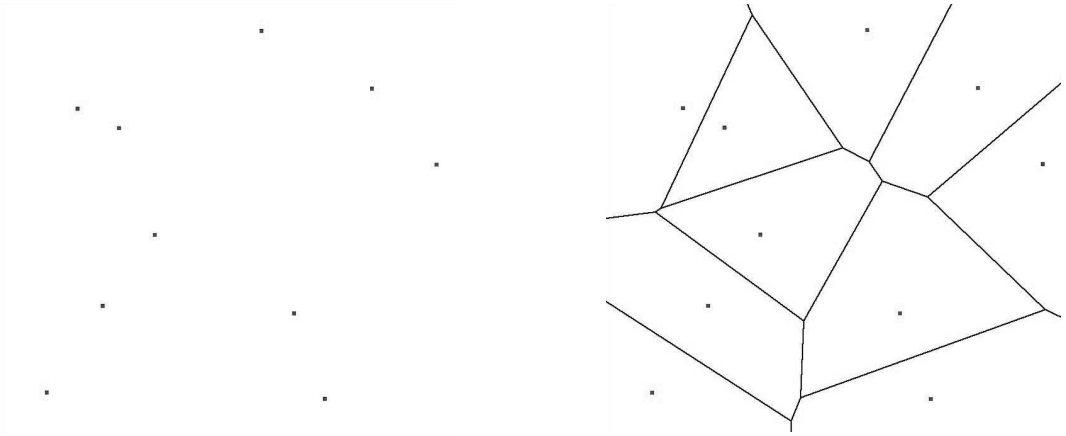


Abbildung 2.12: Punktmenge und dazugehöriges Voronoi-Diagramm

Die Grenzen zwischen zwei Regionen (Voronoizellen) werden als Voronoikanten bezeichnet. Zwei Voronoizellen, die sich durch eine Voronoikante gegenüberliegen, werden als Voronoinachbarn bezeichnet. Hierbei haben alle Punkte, die auf der Voronoikante liegen, denselben euklidischen Abstand zu beiden Zentren. Am Ende einer Voronoikante treffen sich mindestens zwei benachbarte Voronoizellen. Der Punkt, in dem sich die Zellen treffen, wird als Voronoiecke bezeichnet. Eine Voronoiecke bildet den Mittelpunkt eines Kreises, der durch die Zentren aller angrenzenden Voronoizellen geht.

Im Allgemeinen handelt es sich bei einer Voronoi-Zerlegung nicht um Partitionierung des Raumes. Bei einer Partitionierung müssen alle Punkte genau einer Partition zugeordnet sein. Allerdings sind die Punkte, die sich auf einer Voronoikante befinden, nicht eindeutig zugeordnet. Um eine Partitionierung zu erhalten, muss ein zusätzliches eindeutiges Entscheidungskriterium existieren, um die Punkte der Voronoikante eindeutig zuzuordnen. Ein Entscheidungskriterium könnte beispielsweise sein, dass alle Punkte einer Voronoikante der Voronoizelle mit der kleineren x- und y-Koordinate zugeordnet werden.

2.9 Delaunay-Triangulation

Eine Delaunay-Triangulation ist ein Verfahren, um eine Fläche, die aus der konvexen Hülle einer Punktmenge S entsteht, in Dreiecke zu teilen. Die Punkte aus S bilden hierbei jeweils die Eckpunkte der Dreiecke. In Abbildung 2.13 sind eine Punktemenge sowie die dazugehörende Delaunay-Triangulation dargestellt.

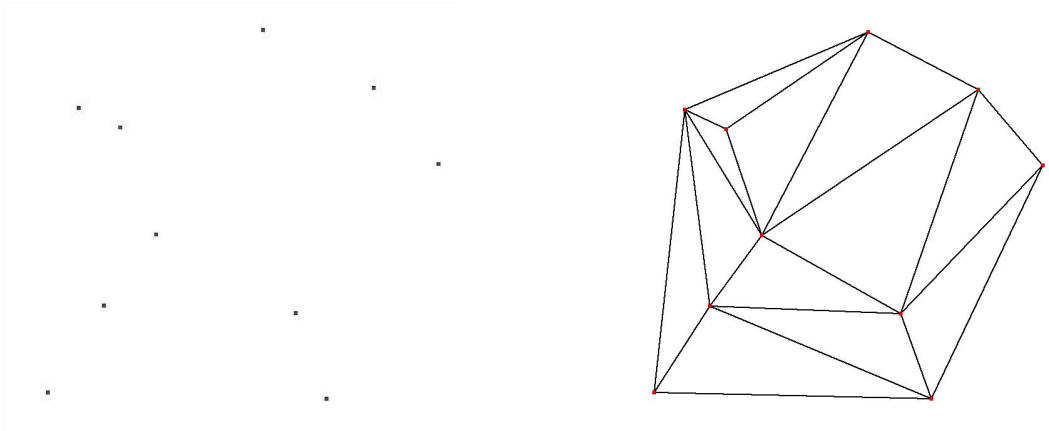


Abbildung 2.13: Punktmenge und dazugehörige Delaunay-Triangulation

In einer Delaunay-Triangulation müssen alle Dreiecke das sogenannte Delaunaykriterium erfüllen [8]. Das Delaunaykriterium beschreibt zwei Anforderungen an die Struktur des Graphen. Die erste Anforderung besagt, dass für jede Kante u, v der Delaunay-Triangulation ein Kreis existieren muss, der durch die Punkte u und v verläuft, aber durch keinen weiteren Punkt aus S. Damit die zweite Anforderung erfüllt wird, darf kein Kreis existieren, der durch die Punkte u, v, w verläuft und zusätzlich noch durch einen Punkt aus $S \backslash \{u, v, w\}$. Werden von allen Dreiecken der Delaunay-Triangulation diese Anforderungen erfüllt, ist die Delaunay-Triangulation eindeutig. Existiert ein Kreis, auf dem mehr als drei Punkte aus S liegen, so existieren mehrere Möglichkeiten die Fläche zu zerlegen. Abbildung 2.14 zeigt ein Beispiel für diesen Fall.

In dieser Zerlegung existiert ein Kreis, der durch vier Punkte aus S verläuft. Dadurch gibt es zwei mögliche Zerlegungen, die durch die gestrichelten Linien dargestellt sind. Der Algorithmus, der im nächsten Abschnitt vorgestellt wird, benötigt ein eindeutiges Kriterium um zu entscheiden, welche Zerlegung verwendet werden soll. Welches Kriterium und somit auch welche Zerlegung verwendet wird, ist von untergeordneter Bedeutung. Entscheidend ist, dass es ein eindeutiges Kriterium gibt.

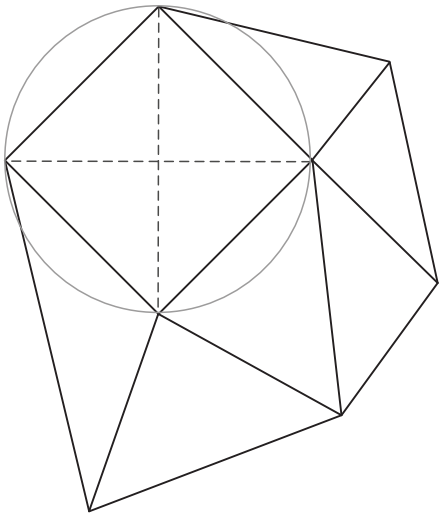


Abbildung 2.14: Nicht eindeutige Delaunay-Triangulation

Die Delaunay-Triangulation einer Punktmenge S verhält sich dual zu dem entsprechenden Voronoi-Diagramm, das durch S bestimmt werden kann. Erstellt man zwischen allen Nachbarn im Voronoi-Diagramm eine Verbindung so erhält man die Delaunay-Triangulation.

2.10 Selbst-stabilisierende und lokale Berechnung eines Delaunaygraphen

In diesem Abschnitt wird ein selbst-stabilisierender und lokaler Algorithmus zur Berechnung eines Delaunaygraphen [42] betrachtet. Dieser Algorithmus ist rundenbasiert und berechnet aus einem initial gegebenen schwach verbundenen Graphen den entsprechenden Delaunaygraphen. Ein schwach verbundener Graph ist ein Graph, in dem jeder Knoten des Graphen von jedem anderen Knoten aus erreicht werden kann, wenn man alle gerichteten Kanten durch ungerichtete Kanten ersetzt. Die Knoten des Graphen sind aus einem zweidimensionalen euklidischen Raum.

Der Algorithmus arbeitet lokal, das bedeutet, dass Knoten nur mit ihren direkt verbundenen Knoten kommunizieren können und auch nur Informationen über diese haben. Die Eingabe des Algorithmus ist ein schwach verbundener Graph $G = (V, E)$ wobei V die Menge der Knoten und E die Menge der Kanten ist. Der Algorithmus arbeitet in Runden und berechnet in jeder Runde ein sogenanntes Delaunay Update $\tilde{G} = (V, \tilde{E})$. Zu diesem Zweck berechnet jeder Knoten $u \in V$ basierend auf den

Knoten, zu denen bereits eine Kante existiert, eine Menge an Kanten $E_S(G, u)$, die für die nächste Runde verwendet werden. Diese Menge besteht aus *stabilen* und *temporären* Kanten:

$$E_S(G, u) = E_{stable}(G, u) \bigcup E_{temp}(G, u)$$

Es existieren zwei Regeln anhand derer entschieden wird, welche Kanten für die nächste Runde erstellt werden. Jeder Knoten berechnet dafür einen aus seiner Sicht gültigen Delaunaygraphen.

Durch die erste Regel werden stabile Kanten erzeugt. Hierbei wird für jedes $v \in V$, das ein Nachbar von u in dessen lokalen Delaunaygraphen ist, eine ungerichtete Kante $\{u, v\}$ in E_{stable} erstellt. Für alle Knoten v, w, die Nachbarn von u sind, wird ebenfalls eine ungerichtete Kante $\{v, w\}$ in E_{stable} erstellt, wenn es keinen Knoten x gibt, der ebenfalls Nachbar von u ist und in dem Kegel liegt, der von den Vektoren $\vec{uv}$ und $\vec{uw}$ aufgespannt wird.

Die zweite Regel erzeugt temporäre Kanten. Hierbei werden alle Knoten h betrachtet, die kein Nachbar im lokalen Delaunaygraphen von u sind. Für jeden Knoten h wird in E_{temp} eine gerichtete Kante zwischen h und dem Nachbarn von u erstellt, der am nächsten an h liegt.

Das vollständige Delaunay Update einer Runde besteht aus allen durch diese beiden Regeln erstellen Kanten:

$$\tilde{E} = \bigcup_{v \in V} E_S(G, v)$$

Die dadurch entstehende Menge $\tilde{E}$ ist ein globaler Schnitt über alle Knoten und kann nicht mit lokalen Wissen gebildet werden. Der Algorithmus terminiert, sobald sich der Graph nicht mehr ändert.

Um also entscheiden zu können, wann der Algorithmus terminiert ist ein globaler Schnitt notwendig. Für die Berechnung des Graphen hingegen, wird nur das lokale Wissen der Knoten verwendet.

2.11 Gears4Net

Gears4Net [75] ist ein asynchrones und nicht-blockierendes Programmiermodell zur Entwicklung von skalierbaren und verteilten Applikationen. Bei der Entwicklung von verteilten Applikationen, wie beispielsweise Applikationen zur Simulation von Peer-to-Peer Systemen, muss die vom Betriebssystem zur Verfügung gestellte Netzwerk API verwendet werden. Diese API bietet blockierende, nicht-blockierende und asynchrone Methoden für die Netzwerkkommunikation.

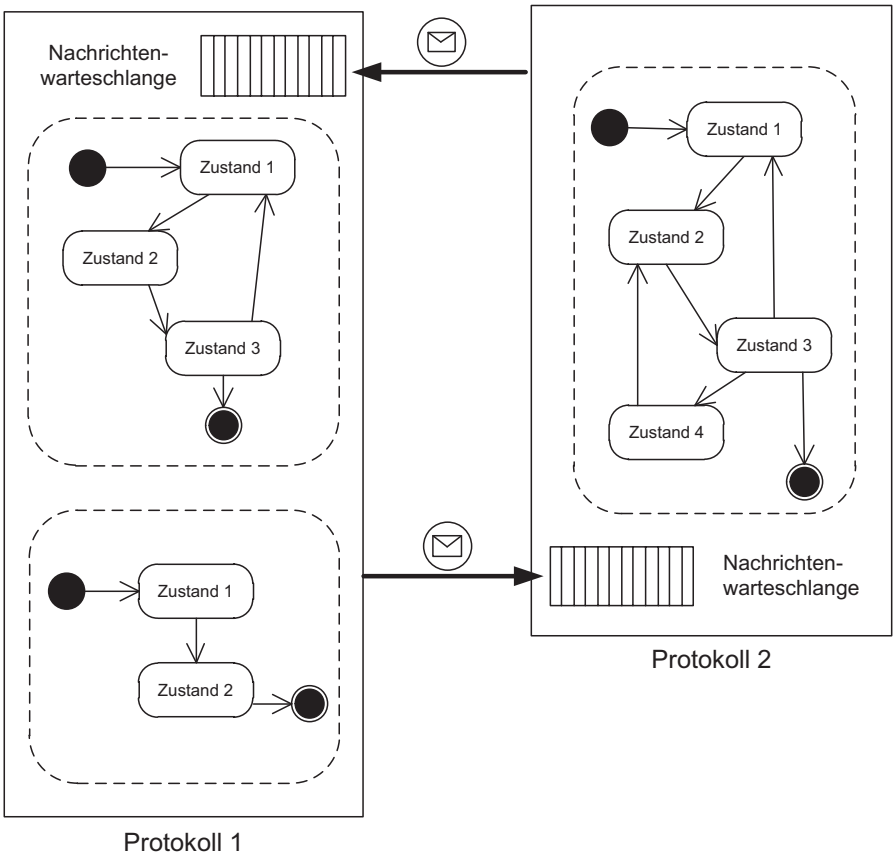


Abbildung 2.15: Gears4Net Protokolle und Zustandsmaschinen [75]

Blockierende Aufrufe sind einfacher und für den Entwickler intuitiver in der Verwendung, da sie einen sequentiellen Programmablauf ermöglichen. Allerdings müssen diese blockierenden Aufrufe häufig in einem eigenen Thread ausgeführt werden, damit nicht die gesamte Anwendung blockiert wird. In der Regel wird so häufig ein Thread pro Verbindung verwendet. Diese erhöhte Anzahl an Threads erhöht die Wahrscheinlichkeit von *Race-Conditions* und den Sychnronisationsaufwand zwischen diesen Threads, wenn auf gemeinsame Ressourcen zugegriffen wird. Zusätzlich wird für jeden Thread ein eigener Programmstack erzeugt. Der Speicherverbrauch

für eine Anwendung, die blockierende Aufrufe verwendet und pro Verbindung einen eigenen Thread verwendet, kann durch

$$Speicherverbrauch := \#Threads \cdot Stackgroesse = \#Verbindungen \cdot Stackgroesse$$

approximiert werden [73]. Daher ist in diesem Programmiermodell die Anzahl möglicher Verbindungen stark abhängig von dem zur Verfügung stehenden Speicher.

Alternativ zu blockierenden Methoden können nicht-blockierende oder asynchrone Methoden verwendet werden. Eine Anwendung, die nicht-blockierende oder asynchrone Methodenaufrufe verwendet, benötigen im Vergleich zu einer Anwendung mit blockierenden Methodenaufrufen weniger Systemressourcen, weil beispielsweise ein einziger Thread für alle aktiven Verbindungen verwendet werden kann. Die Entwicklung ist hierbei allerdings nicht so intuitiv, wie bei blockierenden Aufrufen. Zusätzlich wird die Programmlogik komplexer, da beispielsweise bei asynchronen Methoden mit Rückruffunktionen gearbeitet werden muss.

Gears4Net vereint die Vorteile der verschiedenen Modelle, indem es sie in einem Zustandsmaschinen-basierenden Programmiermodell verbindet. Es erlaubt eine übersichtliche und intuitive Programmstruktur und einen geringen Verbrauch von Systemressourcen, da asynchrone Methodenaufrufe verwendet werden. Gears4Net basiert auf vier Konzepten: Protokolle, Zustandsmaschinen, Scheduler und Wartebedingungen, die im Folgenden kurz erläutert werden.

Eine Gears4Net Anwendung kann aus verschiedenen Protokollen bestehen. Protokolle bestehen aus mehreren Zustandsmaschinen und die Kommunikation zwischen verschiedenen Protokollen basiert auf asynchronem Nachrichtenaustausch (siehe Abbildung 2.15).

Jedem Protokoll ist genau ein Scheduler zugewiesen. Ein Scheduler kann einen oder mehrere Threads verwenden, um eine beliebige Anzahl an Protokollen auszuführen. Hierdurch kann festgelegt werden, wie viele Threads für die Ausführung der Anwendung verwenden werden.

In Gears4Net werden anstatt blockierender Methoden nicht-blockierende Wartebedingungen verwendet. Wartebedingungen in Gears4Net können beliebig komplex werden und aus verschiedenen *Und-* bzw. *Oder-verknüpften* Bedingungen bestehen. Bei diesen Wartebedingungen können auf beliebig definierte Ereignisse, wie beispielsweise der Empfang einer bestimmten Nachricht oder auch auf einen Timeout, gewartet werden. Stellt man ein Gears4Net Protokoll grafisch als Zustandsmaschine dar, entsprechen die Wartebedingungen den Übergängen zwischen den verschiedenen Zuständen (siehe Abbildung 2.15).

Kapitel 3

NAT Traversal

Die Knappheit von IPv4 Adressen und der sehr langsame Umstieg zu IPv6 führen dazu, dass immer noch sehr viele Computer mittels IPv4 und NAT Routern mit dem Internet verbunden sind. Computer, die mittels eines NAT Routers mit dem Internet verbunden sind, können von anderen Computern im Internet nicht direkt kontaktiert werden, da NAT Router eingehende Verbindungen erschweren oder im schlimmsten Fall völlig unmöglich machen. Für Applikationen, die eingehende Verbindungen benötigen wie beispielsweise Peer-to-Peer oder Voice-over-IP Applikationen ist das ein Problem. Um mit diesem Problem umzugehen, werden sogenannte NAT Traversal Verfahren verwendet. Einige dieser Verfahren, wie beispielsweise *UDP Hole Punching*, wurden bereits im vorherigen Kapitel beschrieben.

Im ersten Teil dieses Kapitels werden zunächst das Systemmodell und die in diesem Kapitel verwendete Terminologie beschrieben. Aufbauend darauf wird ein neues Hole Punching Verfahren für TCP vorgestellt, das auf der Injektion eines speziellen TCP Pakets basiert [39]. Im dritten Teil dieses Kapitels werden existierende NAT Traversal Verfahren bezüglich ihrer Anforderungen analysiert sowie ein Regelwerk vorgestellt, mit dessen Hilfe man entscheiden kann, welches Verfahren man in einem konkreten Fall anwenden sollte. Dafür wird ein Protokoll vorgestellt, das die relevanten Eigenschaften der involvierten NAT Router ermittelt [38]. Basierend auf den ermittelten Eigenschaften und der jeweiligen Anforderungen der Verfahren kann das Verfahren ausgewählt werden, das die höchste Erfolgswahrscheinlichkeit bietet.

In Abschnitt 3.4 wird ein Testsystem vorgestellt, mit dem die in diesem Kapitel vorgestellten Verfahren und Protokolle verlässlich evaluiert werden. Der Vorteil des Testsystems ist es, dass durch Netzwerkmitschnitte eindeutig bestimmt werden kann, warum ein Verfahren erfolgreich war bzw. warum es fehlgeschlagen ist. Im anschließenden Abschnitt werden die mit dem Testsystem ermittelten Ergebnisse der getes-

teten Verfahren und Protokolle ausführlich betrachtet. Abschnitt 3.6 fasst abschließend die vorgestellten Arbeiten und Ergebnisse zusammen.

3.1 Systemmodell und Terminologie

In diesem Abschnitt werden das Systemmodell sowie die Terminologie, die im folgenden Kapitel verwendet werden, beschrieben. Es wird davon ausgegangen, dass Computer mittels NAT Routern eine Verbindung mit dem Internet herstellen. Diese NAT Router filtern und modifizieren eingehende und ausgehende Datenpakete gemäß einem festgelegten Regelwerks (siehe Abschnitt 2.4). Des Weiterem wird davon ausgegangen, dass diese Computer in der Lage sind die externe IP Adresse des verwendeten NAT Routers sowie die aktuell verwendete externe Portnummer zu bestimmen. Diese Informationen können beispielsweise mit Hilfe des klassischen STUN Protokolls (siehe Abschnitt 2.5) oder des STUNT Protokolls [30] ermittelt werden.

Um Hole Punching Verfahren durchzuführen, müssen die beteiligen Computer in der Lage sein Informationen über einen gemeinsamen Kommunikationskanal miteinander auszutauschen. Dieser Kommunikationskanal kann mittels eines Servers realisiert werden, der ohne NAT Traversal Mechanismen direkt erreichbar ist und zu dem beide Kommunikationspartner eine Verbindung haben. In Peer-to-Peer Systemen hingegen kann beispielsweise ein Peer, der direkt erreichbar ist (OpenPeer), verwendet werden um Nachrichten zwischen zwei nicht direkt verbundenen Peers auszutauschen. Unabhängig davon, ob ein Server oder OpenPeer verwendet wird, ist der dieser Kommunikationskanal langsamer als eine direkte Verbindung zwischen den beiden Kommunikationspartnern. Des Weiteren kann dieser Kommunikationskanal einen Flaschenhals darstellen, wenn zu viele Computer diesen für ihren dauerhaften Nachrichtenaustausch verwenden würden. Daher sollte dieser Kommunikationskanal nur verwendet werden um eine direkte Verbindung zu initiieren.

Im weiteren Verlauf dieser Arbeit wird die Bezeichnung *SYN Paket* gleichbedeutend mit der Bezeichnung *TCP Paket mit gesetztem SYN Flag* verwendet. Ebenso wird diese Kurzform für TCP Pakete mit gesetztem ACK, SYN/ACK und RST Flag verwendet. Um TCP oder UDP Pakete zu senden oder zu empfangen wird vom Betriebssystem ein Socket zur Verfügung gestellt. Im Folgenden wird der Begriff *aktiver Socket* für einen Socket verwendet, der Daten sendet oder den TCP Verbindungsaufbau initiiert. Analog wird der Begriff *passiver Socket* für einen Socket verwendet, der auf Daten bzw. eine eingehende TCP Verbindung wartet. Der Begriff

Raw Socket beschreibt einen Socket mit dem man seinen eigenen ein- und ausgehenden Netzwerkverkehr mitlesen kann. Um einen Raw Socket erstellen zu können, werden in der Regel Administratorrechte auf dem jeweiligen System benötigt. Der Begriff *Netzwerkumgebung* eines Rechners beschreibt das Vorhandensein eines NAT Routers sowie dessen Eigenschaften wie beispielsweise das Filterverhalten und das Portzuordnungsverhalten.

3.2 SYNI

Ein Standardverfahren, um mit den durch NAT Router entstehenden Verbindungsprobleme umzugehen ist UPnP (siehe Abschnitt 2.6.1). Allerdings hat ein Test von 2700 BitTorrent Nutzern gezeigt, dass trotz solcher Standardverfahren ca. 40% dieser Nutzer Verbindungsprobleme durch die verwendeten NAT Router hatten [44].

Sollte ein Nutzer nicht in der Lage sein eine Portzuordnung manuell oder mittels UPnP zu erstellen, kann man mittels Hole Punching Verfahren versuchen einen direkten Nachrichtenaustausch zu ermöglichen (siehe Abschnitt 2.6.3). Beim Einsatz des verbindungslosen UDP sind diese Verfahren im Allgemeinen simpler und haben eine höhere Erfolgswahrscheinlichkeit im Vergleich zu den Verfahren, die auf TCP basieren. Allerdings benötigten einige Applikationen einen verlässlichen Datentransport. Trotz der Tatsache, dass Protokolle wie RUDP [13] existieren, die einen verlässlichen Datentransport auf UDP bieten, ist es in vielen Fällen besser TCP zu verwenden, da TCP einige Vorteile bietet. Zum einen ist TCP ein standardisiertes und erprobtes Protokoll und zum anderen ist TCP für alle Betriebssysteme verfügbar, in diese integriert und optimiert.

Das Ziel ist es daher, ein TCP Hole Punching Verfahren zu entwickeln, das mit einer sehr hohen Wahrscheinlichkeit eine Verbindung aufbauen kann. Um dieses Ziel zu erreichen ist es notwendig, dass die Anforderungen des Verfahrens mit den Gegebenheiten in der Praxis übereinstimmen. Das Verfahren sollte sich so weit möglich an den standardisierten TCP Nachrichtenfluss halten, weil andererseits Pakete, die nicht dem normalen Nachrichtenfluss folgen, von den NAT Routern verworfen werden könnten. Des Weiteren sollte das Verfahren in der Praxis robust sein und beispielsweise nicht spezielles Zeitverhalten benötigen, da dies in der Praxis nicht kontrolliert werden kann.

Im Folgenden wird ein neues TCP Hole Punching Verfahren namens SYNI vorgestellt. Die Grundidee von SYNI ist es das SYN Paket, das beim Aufbau einer TCP-

Verbindung gesendet wird, über einen separaten Kommunikationskanal vom Quellrechner an den Zielrechner zu senden. Der Zielrechner sendet dann dieses SYN-Paket an seinen eigenen passiven Socket. Das ursprüngliche SYN-Paket, das der Quellrechner gesendet hat, hat den Zielrechner allerdings nie erreicht, da es vom NAT Router des Zielrechners nicht weitergeleitet wurde.

3.2.1 Verfahren

Im folgenden Abschnitt wird das TCP Hole Punching Verfahren SYNI beschrieben, das auf der Injektion eines SYN Pakets basiert. Das Verfahren ist in Abbildung 3.1 grafisch dargestellt. Hierbei versucht Rechner A eine Verbindung mit Rechner B herzustellen. Rechner A verwendet hierbei den NAT Router R_A und Rechner B befindet sich hinter dem NAT Router R_B. Bevor das eigentliche Hole Punching Verfahren durchgeführt wird, müssen A und B jeweils den externen IP Endpunkt bestimmen. Das eigentliche Verfahren wird anschließend in vier Schritten durchgeführt. Damit das Verfahren erfolgreich durchgeführt werden kann, müssen einige Bedingungen seitens der Betriebssysteme der Rechner A und B sowie der verwendeten NAT Router R_A und R_B erfüllt sein. Diese Bedingungen werden im Folgenden mit **(Bx)** nummeriert, da diese Bedingungen im Abschnitt 3.2.2 dafür verwendet werden, verschiedene Hole Punching Verfahren miteinander zu vergleichen.

Im ersten Schritt des Verfahrens erstellt A einen TCP Socket. Der vom Socket verwendete TTL Wert wird so gesetzt, dass die Nachrichten nicht bei R_B ankommen. Zusätzlich erstellt A einen Raw Socket mit dem A seinen eigenen ein- und ausgehenden Netzwerkverkehr mitlesen kann **(B1)**. Anschließend initiiert A mit Hilfe des TCP Sockets den Verbindungsaufbau zu B unter Verwendung des externen IP Endpunkts von B. Hierbei wird von dem TCP Socket ein SYN Paket an B gesendet. A kann mit Hilfe des Raw Sockets die initiale Sequenznummer auslesen, die in dem ausgehenden SYN Paket enthalten ist. Damit das Verfahren erfolgreich durchgeführt werden kann, ist es notwendig, dass R_A die initiale Sequenznummer des SYN Pakets nicht ändert **(B2)**. NAT Router können zum Schutz gegen IP-Spoofing einen zufälligen Offset auf die Sequenznummer addieren (siehe Abschnitt 2.2.3). Wenn das SYN Paket R_A erreicht, erstellt R_A eine Portzuordnung für die Verbindungsanfrage von A und leitet das SYN Paket weiter an R_B. Aufgrund dessen, dass der verwendete Socket einen geringen TTL Wert verwendet, wird das SYN Paket R_B nicht erreichen. Hierdurch wird R_B auch nicht mit einem TCP RST Paket oder einer ähnlichen Fehler Nachricht auf das SYN Paket antworten. Stattdessen empfängt R_A eine ICMP Nachricht vom Typ *Zeitlimit überschritten* von einem Router, der auf dem Pfad von

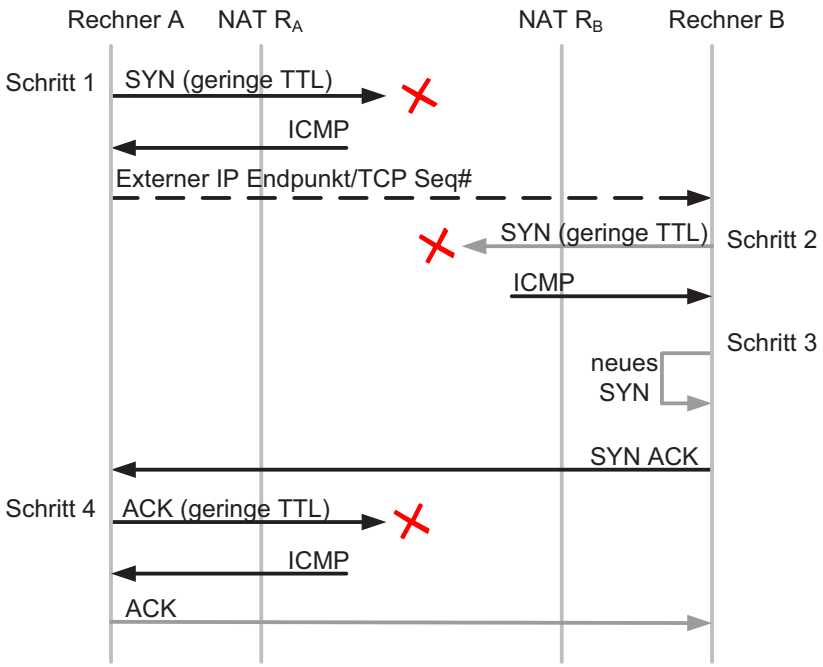


Abbildung 3.1: SYNI - gepunktete Linien sind Nachrichten, die über einen gemeinsamen Kommunikationskanal ausgetauscht werden

R_A nach R_B liegt. Unabhängig davon, ob R_A die ICMP Nachricht an A weiterleitet oder nicht, ist es für eine erfolgreiche Durchführung des Verfahrens notwendig, dass die Portzuordnung bestehen bleibt und der TCP Socket weiterhin auf die zum Verbindungsaufbau zugehörige SYN-ACK Nachricht wartet **(B3)**. Nachdem A die verwendete initiale Sequenznummer ausgelesen hat, sendet A diese Sequenznummer sowie den eigenen externen IP Endpunkt über einen separaten Kommunikationskanal an B.

Im zweiten Schritt erstellt B einen Raw Socket, mit dem eigene TCP Pakete gesendet werden können. B erstellt eine TCP SYN Nachricht mit seinem internen IP Endpunkt als Absender, einem TTL Wert, so dass die Nachricht nicht bei A ankommt, einer zufälligen initialen Sequenznummer und dem externen IP Endpunkt von A als Zieladresse. Anschließend sendet B dieses Paket mit dem Raw Socket und da dieses SYN Paket mit einer geringen TTL gesendet wurde, wird dieses SYN Paket nie bei R_A bzw. A ankommen. Allerdings wurde durch dieses Paket eine Portzuordnung bei R_B erstellt. Analog zum ersten Schritt, wird R_B eine ICMP Nachricht vom Typ *Zeitlimit überschritten* empfangen, welche die vorher erstellte Portzuordnung nicht beeinflussen darf **(B3)**.

Im nächsten Schritt erstellt B einen TCP Socket, der auf einem festgelegten internen IP Endpunkt auf eingehende Nachrichten bzw. Verbindungen wartet. Anschließend erstellt B ein weiteres SYN Paket. B verwendet für die Erstellung der SYN Nachricht die Informationen, die er von A über den gemeinsamen Kommunikationskanal erhalten hat. Das Ziel hierbei ist es ein SYN Paket zu erstellen, das so aussieht wie das Paket, das von A gesendet wurde, aber von R_B gefiltert wurde. Hierfür beinhaltet der TCP Header die externe Portnummer von A als Quellport, die von A im ersten Schritt verwendete initiale Sequenznummer sowie die interne Portnummer, die der passive TCP Socket von B verwendet, als Zielport. Der IP Header des erstellten SYN Pakets beinhaltet die externe IP Adresse von A als Quelladresse und die interne IP Adresse von B als Zieladresse. Das so erstellte TCP/IP Paket muss anschließend mit einem Protokoll der Netzwerkschicht (siehe Abschnitt 2.1.1) wie beispielsweise Ethernet versendet werden. Hierfür erstellt B ein Ethernet Frame und setzt seine eigene Hardware Adresse (MAC Adresse) als Ziel und die Hardware Adresse von R_B als Quelladresse. Das vollständige Paket wird über den Raw Socket von B an dessen passiven TCP Socket gesendet. Nachdem das Paket gesendet wurde, wird der Raw Socket geschlossen. Aufgrund dessen, dass das Paket an den passiven TCP Socket von B gesendet wurde und das Paket so aussieht, als wäre es von A gesendet worden, wird ein neuer Verbindungsstatus erstellt und der TCP Socket von B sendet ein SYN-ACK Paket A. Dieses TCP Paket wird vom Betriebssystem aus über das Netzwerk an R_B gesendet. R_B hat bereits durch das ausgehende SYN Paket eine Portzuordnung für den Nachrichtenfluss erstellt und leitet das Paket weiter in Richtung R_A.

Im vierten und letzten Schritt leitet R_A das SYN-ACK Paket von B weiter an A. Aus der Sicht von R_A gehört dieses SYN-ACK Paket zu dem aus dem ersten Schritt gesendeten SYN Paket. A empfängt dieses SYN-ACK Paket, das zu dem Verbindungszustand der ausgehenden Verbindung zu B passt. Der TCP Socket von A wird daraufhin mit einem ACK Paket antworten. Damit dieses ACK Paket B erreicht, muss A den TTL Wert wieder auf den ursprünglichen Wert zurücksetzen, so dass das Paket nicht unterwegs verworfen wird. Allerdings ist das Umsetzen der TTL während des Verbindungsaufbaus in der Praxis nicht immer problemlos möglich (siehe Abschnitt 3.5.3). Daher setzt A den TTL Wert des TCP Sockets erst um, nachdem dieser das ACK Paket mit geringer TTL gesendet hat, weil zu diesem Zeitpunkt der Verbindungsaufbau für den Socket abgeschlossen ist. A kann dieses ACK Paket mit Hilfe des Raw Sockets aus Schritt 1 mitlesen und es mit Hilfe des Raw Sockets erneut mit der Standard TTL an B senden. R_A leitet beide ACK Pakete an B weiter, allerdings wird nur das zweite ACK Paket R_B erreichen.

Nachdem das ACK Paket von R_B an B weitergeleitet wurde, ist die Verbindung zwischen A und B hergestellt und es können Daten über die verbundenen Sockets miteinander ausgetauscht werden. Aus Sicht von R_A wurde die Verbindung über einen normalen 3-Wege-Handshake herstellt. Die einzige Ausnahme aus Sicht von R_A ist hierbei das doppelte ACK Paket, was allerdings nicht der TCP Spezifikation widerspricht. Aus Sicht von R_B hingegen wurde die Verbindung über eine unübliche Paketsequenz bestehend aus einem ausgehenden SYN und SYN-ACK Paket sowie einem eingehenden ACK Paket **(B4)** hergestellt. In Abschnitt 3.5 wird gezeigt, dass diese Sequenz von den meisten NAT Routern akzeptiert wird.

Standard TTL Das vorgestellte SYNI Verfahren verwendet einen geringen TTL Wert auf der Seite von A, um zu verhindern, dass R_B eine Fehlernachricht sendet. Wenn R_B auf das SYN Paket aus dem ersten Schritt nicht mit einer Fehlermeldung reagiert, so kann dieses Paket mit einem Standard TTL Wert gesendet werden.

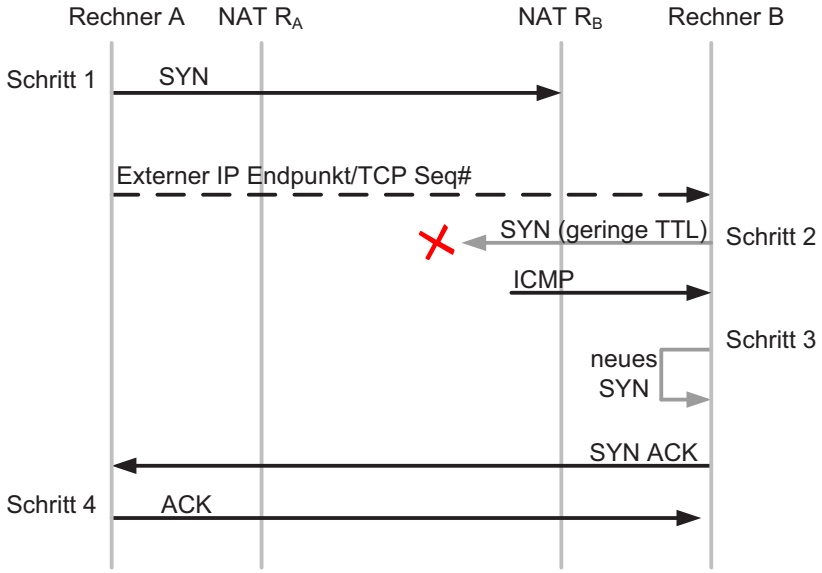


Abbildung 3.2: SYNI mit Standard TTL - gepunktete Linien sind Nachrichten, die über einen gemeinsamen Kommunikationskanal ausgetauscht werden

Hierdurch wird der 4. Schritt des Verfahrens vereinfacht, da das abschließende ACK Paket auch mit dem Standard TTL Wert gesendet wird und somit nicht zusätzlich per Raw Socket gesendet werden muss. Das Verfahren mit einem Standard TTL Wert auf der Seite von A ist in Abbildung 3.2 grafisch dargestellt. In Abschnitt 3.5 werden beide Varianten (geringe und Standard TTL) evaluiert und ihre Erfolgswahrscheinlichkeiten ermittelt.

3.2.2 Verwandte Arbeiten

In diesem Abschnitt werden verwandte Arbeiten im Bereich NAT Traversal und im Speziellen TCP Hole Punching Verfahren beschrieben. Die TCP Hole Punching Verfahren werden hierbei hinsichtlich ihrer Bedingung analysiert und mit SYNI verglichen. In Tabelle 3.1 sind die Bedingung der einzelnen TCP Hole Punching Verfahren zusammengefasst.

Alle TCP Hole Punching Verfahren, die in diesem Abschnitt beschrieben werden, erfordern, dass A und B in der Lage sind über einen gemeinsamen Kommunikationskanal den Verbindungsaufbau zu koordinieren und für den Verbindungsaufbau relevante Informationen, wie beispielsweise die externen IP Endpunkte, auszutauschen.

NUTSS

Guha et al. beschreiben in [33] eine Architektur namens NUTSS, die zwei verschiedene TCP Hole Punching Verfahren beinhaltet. Das erste Verfahren ist in Abbildung 3.3 (a) grafisch dargestellt und benötigt einen sogenannten STUNT Server, der in der Lage ist Pakete mit falscher Absenderadresse (IP-Spoofing) zu versenden. Die beiden Rechner A und B initiieren zeitgleich einen Verbindungsaufbau zueinander unter Verwendung eines TCP Sockets. Hierbei wird von beiden TCP Sockets jeweils ein SYN Paket erstellt und an den jeweiligen Kommunikationspartner gesendet. Dieses SYN Paket wird jeweils von A und B mittels eines Raw Sockets **(B1)** mitgelesen und die Informationen dieses SYN Pakets werden anschließend an den STUNT Server übertragen. Hierbei ist es notwendig, dass die NAT Router R_A und R_B die TCP Sequenznummern nicht ändern **(B2)**. Sollten die NAT Router die Sequenznummern ändern, wären die Informationen, die an den STUNT Server gesendet wurden, ungültig und das Verfahren kann nicht erfolgreich durchgeführt werden. Eine weitere Bedingung ist, dass die SYN Pakete nicht beim Ziel NAT Router ankommen, weil diese sonst ein TCP RST Paket erzeugen könnten. Daher werden die SYN Pakete mit einem geringen TTL Wert gesendet. Die beiden NAT Router werden daraufhin eine ICMP Nachricht vom Typ *Zeitlimit überschritten* empfangen. Für eine erfolgreiche Durchführung von NUTSS (a) ist es notwendig, dass diese ICMP Nachrichten die Portzuordnung der NAT Router nicht beeinflussen **(B3)**.

Sobald der STUNT Server die Nachrichten mit den Informationen der SYN Pakete erhalten hat, erstellt der STUNT Server jeweils ein SYN-ACK Paket, das so aussieht, als wäre es von A bzw. B gesendet worden. R_A und R_B werden diese SYN-ACK

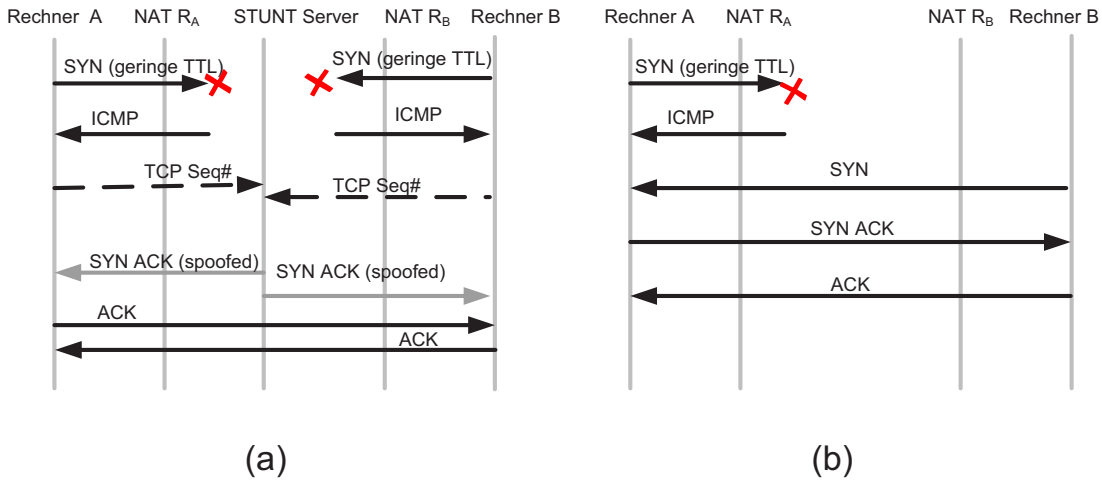


Abbildung 3.3: NUTSS [32] - gepunktete Linien sind Nachrichten, die an einen STUNT Server gesendet werden

Pakete an A und B weiterleiten, da diese Pakete dem erwarteten Paketfluss entsprechen. Nachdem die TCP Sockets von A und B mit dem abschließenden ACK Paket geantwortet haben, ist die Verbindung erfolgreich aufgebaut.

Die Tatsache, dass dieser Ansatz auf dem Versenden von Paketen mit gefälschter Absenderadresse basiert **(B5)**, ist ein signifikanter Nachteil, da Internet Service Provider (ISPs) *Egress Filter* verwenden können, um ihr Netzwerk vor Paketen mit gefälschter Absenderadresse zu schützen (siehe Abschnitt 2.2.3). Kommen die Pakete des STUNT Servers aufgrund des *Egress Filters* nicht bei A oder B an, kann mit diesem Ansatz keine Verbindung hergestellt werden.

Das zweite Verfahren, das in [33] vorgestellt wurde (siehe Abbildung 3.3 (b)) basiert nicht, wie der vorherige Ansatz, auf dem Versenden von Paketen mit gefälschter Absenderadresse. Dieser Ansatz benötigt weder einen STUNT Server noch Raw Sockets.

A initiiert mit einem TCP Socket einen Verbindungsaufbau zu B. Hierbei wird, wie schon beim vorherigen Ansatz, ein geringer TTL Wert verwendet. Das SYN Paket erreicht B bzw. R_B nicht, allerdings erstellt R_A die Portzuordnung und empfängt anschließend eine ICMP Nachricht von Typ *Zeitlimit überschritten*. Die empfangene ICMP Nachricht darf keinen Einfluss auf die Portzuordnung haben **(B3)**, weil das Verfahren sonst nicht erfolgreich durchgeführt werden kann. Unmittelbar nachdem A den Verbindungsaufbau initiiert hat und das SYN Paket an B gesendet wurde, wird der verwendete Socket geschlossen und ein neuer Socket erstellt, der dieselbe Portnummer verwendet wie der vorherige Socket, aber auf eine eingehende Verbin-

dung wartet. Hierbei wird vorausgesetzt, dass der Socket, wenn er geschlossen wird, kein TCP RST Paket sendet **(B6)**, da dieses Paket Auswirkungen auf die Portzuordnung von R_A haben kann. Nachdem A den neuen Socket erstellt hat, initiiert B einen Verbindungsaufbau zu A. Die Verbindung von B zu A wird erfolgreich hergestellt, wenn R_A den nicht konformen Paketfluss ausgehendes SYN Paket, eingehendes SYN Paket zulässt **(B8)**.

P2PNAT

Ford et al. haben in [27] ein TCP NAT Traversal Verfahren (P2PNAT) vorgestellt, das auf dem TCP *Simultanes Öffnen* Szenario basiert (siehe Abschnitt 2.1.4). A und B erstellen jeweils einen aktiven und einen passiven TCP Socket. Beide Sockets werben hierbei an dieselbe Portnummer gebunden. Normalerweise kann eine Portnummer nur an einen aktiven oder passiven TCP Socket gebunden werden, aber nicht an mehrere Sockets. Damit sich mehrere TCP Sockets an dieselbe Portnummer binden können, muss eine spezielle Socket Option namens *SO_REUSEADDR* **(B7)** verfügbar sein. Diese Option wird von allen gängigen Betriebssystemen unterstützt [27].

Um eine Verbindung aufzubauen, initiieren beide Rechner zur gleichen Zeit einen Verbindungsaufbau zueinander, wie in Abbildung 3.4 (a) dargestellt. Durch die beiden ausgehenden TCP SYN Pakete werden von R_A und R_B die entsprechenden Portzuordnungen erstellt. Wenn es A und B gelingt, den Verbindungsaufbau zeitlich so zu initiieren, dass die jeweilige Portzuordnung bereits erstellt ist, bevor das SYN Paket des anderen Rechners den NAT Router erreicht, können die Router die Pakete weiterleiten. Damit R_A und R_B die Pakete in das interne Netzwerk weiterleiten, müssen sie die Paketsequenz ausgehendes SYN Paket, eingehendes SYN Paket **(B8)** akzeptieren. Nachdem A und B die SYN Pakete empfangen haben, wird die Verbindung durch die SYN-ACK sowie die darauffolgenden ACK Pakete erfolgreich hergestellt.

Die Herausforderung bei der Durchführung dieses Verfahrens liegt darin die Verbindung auf beiden Seiten zur gleichen Zeit zu initiieren. Sollte das nicht gelingen, würde ein SYN Paket den Ziel NAT Router erreichen, bevor dieser die Portzuordnung erstellt hat. Das könnte zur Folge haben, dass eine entsprechende Fehlernachricht vom Ziel NAT Router erzeugt wird. Diese zeitliche Abhängigkeit ist durch verschiedene Latenzen und Schwankungen im Netzwerk in der Praxis sehr schwer kontrollierbar.

In Abhängigkeit davon, ob und wie die involvierten Betriebssysteme das *Simultane Öffnen* gemäß der TCP Spezifikation [63] in Verbindung mit der Socket Option

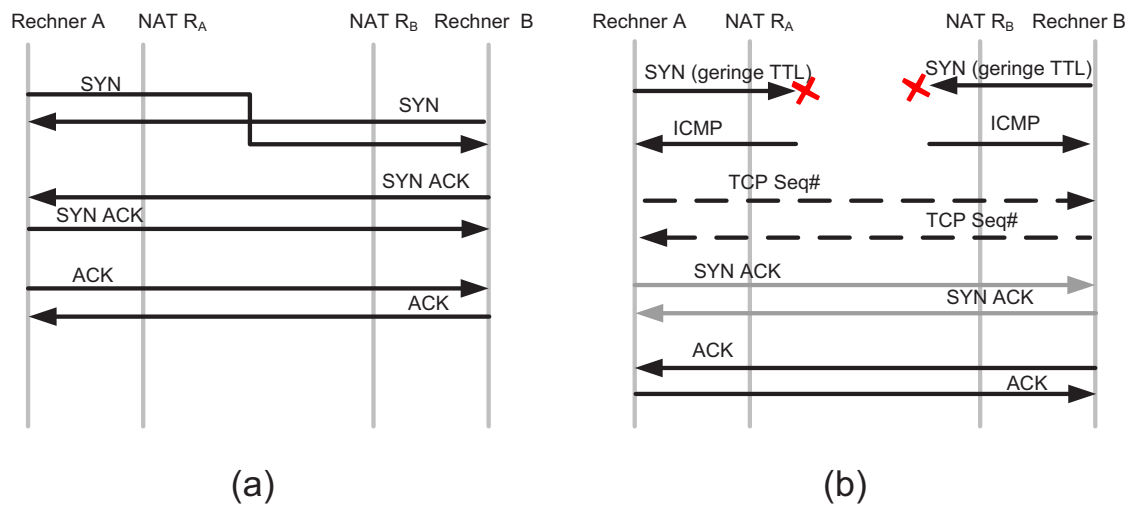


Abbildung 3.4: P2PNAT und NATBLASTER [32] - gepunktete Linien sind Nachrichten, die über einen gemeinsamen Kommunikationskanal ausgetauscht werden

SO_REUSEADDR implementiert haben, kann dies zu unterschiedlichem Verhalten bei den Sockets führen. Hierbei kann beispielsweise entweder nur der aktive oder der passive Socket eine erfolgreiche Verbindung melden oder sogar beide Sockets. In diesem Fall müsste die Applikation mit den verschiedenen Verhaltensweisen umgehen können.

NATBLASTER

In [11] wurde ein TCP Hole Punching Verfahrens namens NATBLASTER vorgestellt. Dieses Verfahren ist vergleichbar mit dem Verfahren NUTSS wie in Abbildung 3.3 (a) dargestellt. Allerdings basiert es nicht auf dem Versenden von Paketen mit gefälschter Absenderadresse.

A und B initiieren einen Verbindungsaufbau mit einem TCP Socket und einem geringen TTL Wert, wie in Abbildung 3.4 (b) dargestellt. Beide Kommunikationspartner lesen das ausgehende SYN Paket mit einem Raw Socket **(B1)** mit und tauschen die Informationen über das ausgehende SYN Paket über den gemeinsamen Kommunikationskanal miteinander aus. Damit diese Information verwendet werden kann, ist es notwendig, dass R_A und R_B die Sequenznummer des ausgehenden SYN Pakets nicht ändern **(B2)**. Eine weitere Bedingung für die erfolgreiche Durchführung des Verfahrens ist, dass die Portzuordnungen von R_A und R_B nicht von der ICMP Nachricht vom Typ *Zeitlimit überschritten* beeinflusst werden **(B3)**. Im Gegensatz zum Verfahren P2PNAT erreichen die SYN Pakete bei NATBLASTER nicht den Ziel NAT

Verfahren	B1		B2		B3		B4		B5	B6		B7		B8	
	A	B	A	B	A	B	A	B		A	B	A	B	A	B
NUTSS (a)	✓	✓	✓	✓	✓	✓	-	-	✓	-	-	-	-	-	-
NUTSS (b)	-	-	-	-	✓	-	-	-	-	✓	-	-	-	✓	-
P2PNAT	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓
NATBLASTER	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-
SYNI	✓	✓	✓	✓	✓	✓	-	✓	-	-	-	-	-	-	-

B1 RAW Socket
B2 Keine Änderung der TCP Sequenznummer
B3 Unbeeinflusst von *ICMP Zeitlimit überschritten*
B4 Ausgehendes SYN und SYN-ACK Paket, eingehendes ACK Paket
B5 IP Spoofing
B6 Kein TCP RST beim Schließen des Sockets
B7 Socket Option SO_REUSEADDR
B8 Ausgehendes SYN Paket, eingehendes SYN Paket

Tabelle 3.1: Bedingungen für TCP Hole Punching Verfahren

Router, so dass die TCP Sockets kein SYN-ACK Paket senden. Stattdessen erstellen A und B mit Hilfe der Informationen der ausgetauschten SYN Pakete die entsprechenden SYN-ACK Pakete manuell und senden diese per Raw Socket zueinander. Nachdem A und B die SYN-ACK Pakete empfangen haben, antworten die TCP Sockets mit den entsprechenden ACK Paketen und die Verbindung ist erfolgreich aufgebaut.

Die Bedingungen, die für einen erfolgreichen Verbindungsaufbau mit NATBLASTER erfüllt sein müssen, sind mit denen von SYNI vergleichbar. Beide Verfahren setzen voraus, dass ein ausgehendes SYN-ACK Paket erfolgreich an den Kommunikationspartner gesendet werden kann, ohne dass zuvor ein eingehendes SYN Paket empfangen wurde und dass anschließend noch ein ACK Paket empfangen werden kann **(B4)**.

Der Unterschied zum Verfahren SYNI ist, das NATBLASTER dieses Verhalten bei beiden involvierten NAT Routern benötigt und SYNI benötigt dieses Verhalten nur bei R_B. In Abschnitt 3.5 wird gezeigt, dass Router existieren, die diese Paketsequenz nicht akzeptieren, so dass mit NATBLASTER in diesen Fällen keine Verbindung aufgebaut werden kann.

Weitere

Zusätzlich zu den in den vorherigen Abschnitten vorgestellten TCP Hole Punching Verfahren existiert eine Reihe weiterer Arbeiten [56, 79, 88] im Bereich NAT Traversal Verfahren. Da es sich bei diesen Verfahren um keine TCP Hole Punching Verfahren handelt, werden diese Verfahren nicht ausführlich beschrieben und auch nicht mit dem SYNI Verfahren und dessen Anforderungen verglichen.

3.3 Protokoll zur Bestimmung der Netzwerkumgebung

Moderne Dienste wie beispielsweise Internettelefonie (Voice-Over-IP) oder Peer-to-Peer-basierte Systeme setzen voraus, dass die Rechner, die an diesen Diensten teilnehmen, in der Lage sind Verbindungen untereinander aufzubauen. Um dies in Verbindung mit NAT Routern zu gewährleisten wurden NAT Traversal Verfahren entwickelt, von denen einige in den letzten Abschnitten näher betrachtet wurden.

Jedes dieser NAT Traversal Verfahren stellt Bedingungen an die eingesetzten NAT Router und Betriebssysteme, die erfüllt sein müssen, damit beispielsweise eine TCP Verbindung hergestellt werden kann. In [32, 34] wurden verschiedene NAT Router auf für NAT Traversal Verfahren relevante Eigenschaften getestet. Beide Arbeiten zeigen, dass die Eigenschaften der analysierten NAT Router variieren. Die Wahrscheinlichkeit für eine erfolgreiche Ausführung eines NAT Traversal Verfahrens ist somit stark abhängig von den Eigenschaften der eingesetzten NAT Router und Betriebssysteme. Makinen et al. haben in [53] einige TCP NAT Traversal Verfahren in Mobilfunknetzen [53] getestet. Auch hierbei konnten für die eingesetzten NAT Router und Firewalls gezeigt werden, dass die Eigenschaften variieren, was zu unterschiedlichen Erfolgswahrscheinlichkeiten der getesteten TCP NAT Traversal Verfahren führte. Um erfolgreich einen Nachrichtenaustausch zwischen zwei Rechnern zu ermöglichen, können entweder alle NAT Traversal Verfahren durchprobiert werden oder man versucht das Verfahren auszuwählen, was die höchste Erfolgswahrscheinlichkeit bietet.

Um das Verfahren mit der höchsten Erfolgswahrscheinlichkeit auswählen zu können, wurde im Rahmen dieser Arbeit das *Mapping Filtering Behavior* (MFB) Protokoll entwickelt. Zu diesem Zweck ermittelt das MFB Protokoll eine Reihe von Eigenschaften der beim NAT Traversal eingesetzten NAT Router. Basierend auf

den ermittelten Eigenschaften wird mit Hilfe eines Regelwerks das Verfahren mit der höchsten Erfolgswahrscheinlichkeit ausgewählt. Die Basis des MFB Protokolls bildet das klassische STUN Protokoll (siehe Abschnitt 2.5). Im Gegensatz zum klassischen STUN Protokoll verwendet das MFB Protokoll zur Beschreibung des grundlegenden Verhaltens eines NAT Routers, die in Abschnitt 2.4 beschriebene Klassifikation. Zusätzlich zum grundlegenden Verhalten ermittelt das MFB Protokoll weitere Eigenschaften. Die Auswahl der Eigenschaften, die vom MFB Protokoll ermittelt werden, wurden durch eine Analyse existierender NAT Traversal Verfahren bestimmt (siehe Abschnitt 3.3.1). Das MFB Protokoll ermittelt beispielsweise die Verfügbarkeit von UPnP und überprüft, ob der NAT Router die TCP Sequenznummern modifiziert.

Die bekanntesten Anwendungen, die eingehende Verbindungen benötigen, sind Peer-to-Peer-basierte Anwendungen. Daher wurde das MFB Protokoll so entwickelt, dass es ohne Probleme in dezentralen Systemen eingesetzt werden kann. Im Allgemeinen wird eine Reihe von Servern verwendet, die das MFB Protokoll ausführen und so den MFB Dienst zur Verfügung stellen. In Verbindung mit einem Peer-to-Peer System ist es möglich, dass einige der teilnehmenden Rechner (Peers) die Aufgaben der Server übernehmen. Hierdurch kann die Notwendigkeit von dedizierten MFB Servern reduziert werden. Je mehr Peers dem Peer-to-Peer Netzwerk beitreten und somit den MFB Dienst in Anspruch nehmen, desto mehr Peers können auch den MFB Dienst anbieten. Die Möglichkeit dass Peers den MFB Dienst zur Verfügung stellen können, erlaubt es ein vollständig dezentrales Peer-to-Peer Netzwerk zu betreiben, ohne dass dedizierte MFB Server benötigt werden.

Im folgenden Abschnitt werden die Eigenschaften, die durch das MFB Protokoll ermittelt werden, beschrieben. Die Ermittlung dieser Eigenschaften dient hierbei als Anforderung, die vom MFB Protokoll erfüllt werden muss. Anschließend wird in Abschnitt 3.3.2 das MFB Protokoll im Detail vorgestellt. In Abschnitt 3.3.3 wird beschrieben, wie das MFB Protokoll in einem dezentralen System eingesetzt werden kann. Die Auswahl des zu verwendenden NAT Traversal Verfahrens basierend auf den durch das MFB Protokoll ermittelten Eigenschaften wird in Abschnitt 3.3.4 näher betrachtet. Abschließend werden in Abschnitt 3.3.5 verwandte Arbeiten beschrieben und mit dem MFB Protokoll verglichen.

3.3.1 Anforderungen

In diesem Abschnitt werden die Eigenschaften der NAT Router, die durch das MFB Protokoll bestimmt werden müssen, näher betrachtet.

Das *Internet Gateway Device* (IGD) Protokoll, das Teil des *Universal Plug and Play* (UPnP) Protokolls ist, ermöglicht es programmatisch eine Portweiterleitung zu erstellen. Die erste Anforderung besteht darin, zu überprüfen, ob per *UPnP* eine Portweiterleitung erstellt werden kann.

Wenn der NAT Router kein UPnP unterstützt, so kann eine direkte Verbindung aufgebaut werden, wenn einer der NAT Router ein Endpunkt-unabhängiges Filterverhalten hat. Daher ist das *Filterverhalten* des NAT Routers die nächste Eigenschaft, die bestimmt wird.

Sollte keiner der beiden Rechner aufgrund der jeweiligen Netzwerkumgebungen eingehende Verbindungen erlauben, können gegebenenfalls verschiedene Hole Punching Verfahren eingesetzt werden. In Abschnitt 3.2.2 wurden eine Reihe von TCP Hole Punching Verfahren sowie die jeweiligen Anforderungen an die Netzwerkumgebung beschrieben.

Hole Punching Verfahren benötigen, um erfolgreiche eine Verbindung aufzubauen, die externen IP Endpunkte der beteiligten Kommunikationspartner. Zu diesem Zweck muss das *Portzuordnungsverhalten* bestimmt werden. Sollte der NAT Router kein Endpunkt-unabhängiges Portzuordnungsverhalten haben, ist es eventuell möglich die Portnummer vorherzusagen, die als nächstes vom NAT Router verwendet wird. Zu diesem Zweck wird das *Portauswahlverhalten* eines NAT Router ermittelt. Eine zusätzliche Eigenschaft, die in diesem Zusammenhang ebenfalls ermittelt werden muss, ist die Eigenschaft der Porterhaltung. Bei der Porterhaltung versucht ein NAT Router die vom lokalen Rechner verwendete interne Portnummer der gleichen externen Portnummer zuzuordnen. Die Information darüber, ob ein NAT Router die Porterhaltung unterstützt wird für einen direkten Verbindungsaufbau in Verbindung mit einem Port-abhängigen Filterverhalten benötigt (siehe Abschnitt 3.3.4).

Einige der in Abschnitt 3.2.2 vorgestellten Verfahren senden Pakete mit einem geringen TTL Wert, um beispielsweise zu verhindern, dass ein TCP RST Paket vom Ziel NAT Router gesendet wird. Allerdings ist das Modifizieren des TTL Wertes nicht immer problemlos möglich, wie in Abschnitt 3.5 gezeigt wird. Aufgrund dessen, dass nicht alle NAT Router eine Antwort auf ein gefiltertes Paket senden, ist es in einigen Fällen nicht notwendig eine Nachricht mit geringem TTL Wert zu senden. Damit nur in den Fällen eine Nachricht mit geringem TTL Wert gesendet werden muss, in denen dies auch notwendig ist, muss das *Filterantwortverhalten* eines NAT Routers bestimmt werden.

Die TCP Hole Punching Verfahren SYNI und NATBLASTER (siehe Abschnitt 3.2.2) benötigen die TCP Sequenznummer des initialen SYN Pakets. Um diese bestimmen

zu können, wird ein spezieller *Raw Socket* benötigt. Daher muss bestimmt werden, ob ein solcher Socket zur Verfügung steht. Steht ein solcher Socket zur Verfügung kann anschließend ermittelt werden, ob der NAT Router die *TCP Sequenznummern modifiziert*.

Das Verfahren NUTSS (b) setzt voraus, dass der Socket *kein TCP RST sendet, nachdem der Socket geschlossen wurde*, da andernfalls eine zuvor erstellte Portzuordnung wieder entfernt werden könnte. Daher muss überprüft werden, wie sich der Socket verhält, wenn dieser geschlossen wird.

Einige der in Abschnitt 3.2.2 vorgestellten Verfahren sowie das Verfahren SYNI benötigten eine spezielle Sequenz von TCP Paketen, die nicht vom NAT Router gefiltert (TCP Zustandsfilterung) werden darf. Eine exakte Bestimmung der TCP Zustandsfilterung in Kombination mit den auftretenden Fehlernachrichten (TCP RST oder ICMP) ist allerdings aufwändig. Zusätzlich müsste sichergestellt werden, dass die Fehlernachrichten nicht in einer anderen Reihenfolge auftreten, weil hierdurch das ermittelte Verhalten beeinflusst werden könnte. Aufgrund dessen, wird die TCP Zustandsfilterung nicht ermittelt.

Die bisher herausgestellten Eigenschaften können zwischen den verschiedenen Transportprotokollen variieren. Dieses unterschiedliche Verhalten betrifft, abgesehen von den TCP spezifischen Eigenschaften wie beispielsweise der TCP Sequenznummern Modifikation, vor allem das Portzuordnungs-, Portauswahl- und Filterverhalten sowie das Filterantwortverhalten. Aus diesem Grund ist eine weitere Anforderung, dass die Eigenschaften *getrennt für TCP und UDP ermittelt* werden müssen.

Die letzte zu ermittelnde Eigenschaft ist die Unterstützung für *Hairpinning*. Hairpinning [27] beschreibt die Eigenschaft eines NAT Routers Pakete zwischen zwei internen Rechnern weiterzuleiten, wobei die Pakete jeweils an die externen IP Endpunkte der Rechner adressiert sind. Dieser Fall kann eintreten, wenn Rechner A über einen Rendezvous Server im öffentlichen Netzwerk den externen IP Endpunkt von B erhalten hat. Versucht A eine Verbindung zu B aufzubauen und der NAT Router unterstützt Hairpinning, werden die Pakete intern weitergeleitet. Unterstützt der NAT Router hingegen kein Hairpinning so ist es möglich, dass keine Verbindung zustande kommt. Obwohl Hairpinning kein direktes NAT Traversal Verfahren ist, so erfordert die fehlende Unterstützung eine spezielle Behandlung in der Applikation, so dass beispielsweise der interne IP Endpunkt zusätzlich bekannt gemacht wird.

Im Folgenden sind die Eigenschaften, die vom MFB Protokoll bestimmt werden müssen, zusammengefasst.

A1: Verfügbarkeit von UPnP

A2: Filterverhalten

A3: Portzuordnungsverhalten

A4: Porterhaltung

A5: Portauswahlverhalten

A6: Filterantwortverhalten

A7: TCP Sequenznummern Modifikation

A8: Verfügbarkeit von Raw Sockets

A9: TCP RST nachdem der Socket geschlossen wurde

A10: Separate Ergebnisse für TCP und UDP

A11: Unterstützung von Hairpinning

3.3.2 Protokoll

Nachdem im vorherigen Abschnitt die Anforderungen für das MFB Protokoll spezifiziert wurden, wird in diesem Abschnitt das MFB Protokoll vorgestellt. Ein Rechner, der seine Netzwerkumgebung bestimmen möchte, wird im Folgenden als MFB Client bezeichnet. Dieser MFB Client kommuniziert hierbei mit einem MFB Server mit einem Anfrage/Antwort Protokoll. Die Aufgabe des MFB Servers besteht darin den MFB Client bei der Bestimmung seiner Netzwerkumgebung zu unterstützen. Ein MFB Server arbeitet zustandslos und muss über zwei externe IP Adressen (X, Y) und über jeweils zwei externe Portnummern (x, y) direkt erreichbar sein. Insgesamt ist ein MFB Server über die IP Endpunkte *X:x*, *X:y*, *Y:x* und *Y:y* erreichbar.

Ein MFB Client, der seine Netzwerkumgebung bestimmen möchte, führt das MFB Protokoll in drei Phasen aus, wie in Abbildung 3.5 dargestellt. In der ersten Phase wird überprüft, ob ein UPnP-fähiger NAT Router im lokalen Netzwerk existiert. Wenn ein UPnP-fähiger NAT Router existiert, kann der MFB Client eine externe Portzuordnung erstellen und das Protokoll terminiert. Existiert kein UPnP-fähiger NAT Router, wird die zweite Phase des Protokolls ausgeführt. Um diese Phase auszuführen muss der MFB Client einen direkt erreichbaren IP Endpunkt eines MFB Servers kennen. Die zweite Phase des MFB Protokolls umfasst das Filterverhalten, das Filterantwortverhalten sowie den Socket- und TCP Sequenznummerntest. In

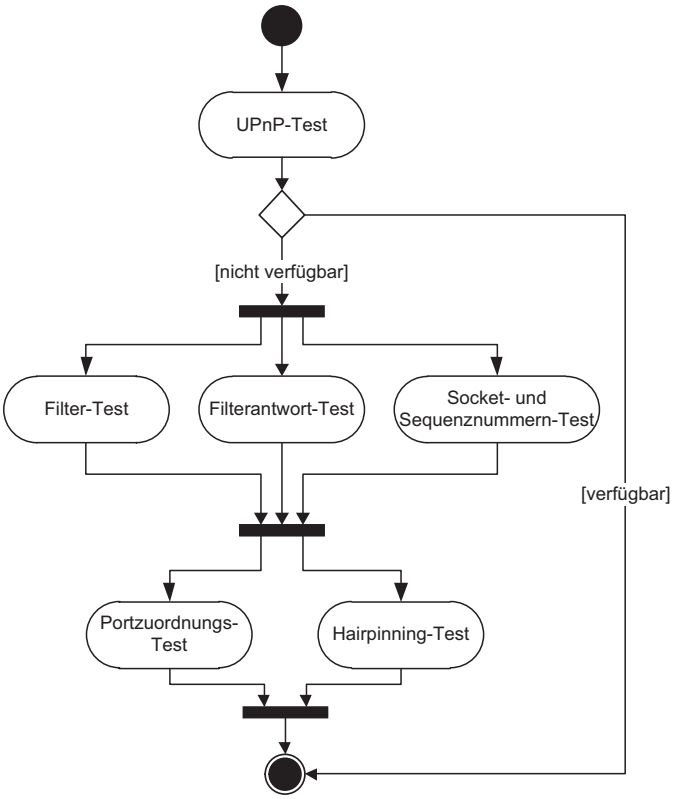


Abbildung 3.5: Phasen des MFB Protokolls

der dritten und letzten Phase wird das Portzuordnungsverhalten bestimmt sowie die Unterstützung für *Hairpinning* ermittelt.

Im Folgenden werden die im MFB Protokoll verwendeten Nachrichtenformate sowie die drei Phasen im Detail beschrieben.

Nachrichten

Das Nachrichtenformat des MFB Protokolls basiert auf dem Nachrichtenformat des klassischen STUN Protokolls. Alle Nachrichten bestehen aus einem 20 Byte großen Header, der aus einem 2 Byte großen Nachrichtentyp, einer 2 Byte großen Nachrichtenlänge sowie einer 16 Byte großen Transaktionsnummer besteht. Das MFB Protokoll verwendet zwei Nachrichtentypen des klassischen STUN Protokolls. Der erste Nachrichtentyp ist der *BindingRequest*, der vom Client an den Server gesendet wird und der zweite Nachrichtentyp ist der *BindingResponse*, der als Antwort vom Server an den Client gesendet wird.

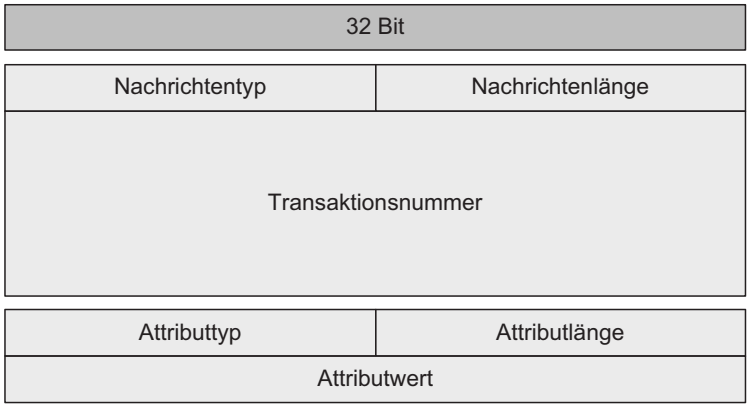


Abbildung 3.6: MFB Nachrichten Header

Nach dem Nachrichtenheader folgt eine variable Anzahl an Attributen. Jedes Attribut besteht aus einem 2 Byte großen Attributtyp, einer 2 Byte großen Attributlänge sowie einem beliebigen Attributwert, wie in Abbildung 3.6 dargestellt. Attribute sind gemäß der Spezifikation des klassischen STUN Protokolls entweder *gefordert* oder *optional*. Das MFB Protokoll verwendet dieselbe Menge an geforderten Attributen und ergänzt das klassische STUN Protokoll um einige neue optionale Attribute. Durch die Verwendung von optionalen Attributen kann ein MFB Client mit MFB Servern und auch mit klassischen STUN Servern kommunizieren. Empfängt ein klassischer STUN Server ein optionales Attribut, das nicht der klassischen STUN Spezifikation entspricht, wird dieses Attribut vom Server ignoriert. Die Kompatibilität zum klassischen STUN Protokoll ermöglicht es eine reduzierte Version des MFB Protokolls mit den klassischen STUN Servern im Internet durchzuführen, für den Fall, dass keine MFB Server verfügbar sind.

Eine *BindingRequest* Nachricht kann die folgenden Attribute beinhalten:

- *Change_Request* (gefordert): Der Wert dieses Attributes umfasst die beiden Flags *ChangeIP* und *ChangePort*. Diese beiden Flags geben wie beim klassischen STUN Protokoll an, über welchen öffentlichen IP Endpunkt der Server seine Antwort Nachricht senden soll.

- *FirewallResponse_Request* (optional): Dieses neue Attribut weist den Server an, das Antwortverhalten von nicht zugeordneten oder gefilterten Portnummern zu ermitteln.

- *SequenceNumber_Request* (optional): Hierbei wird der Server angewiesen, die TCP Sequenznummer der Anfragenachricht des MFB Clients auszulesen und diese mit der Antwortnachricht an den Client zurückzusenden.

Eine *BindingResponse* Nachricht, die vom MFB Server gesendet wird, kann die folgenden Attribute beinhalten:

- *Mapped_Address* (gefordert): Dieses Attribut beinhaltet den externen IP Endpunkt mit dem der Client mit dem Server kommuniziert.
- *Changed_Address* (gefordert): Dieses Attribut beinhaltet den alternativen IP Endpunkt des MFB Servers, der verwendet wird, wenn das Flag *ChangeIP* und *ChangePort* gesetzt ist.
- *Source_Address* (gefordert): Dieses Attribut gibt die Absenderadresse des MFB Servers an. Der MFB Client kann hierdurch feststellen, ob sich der MFB Server hinter einem weiteren NAT Router befindet. In diesem Fall könnten die ermittelten Ergebnisse fehlerhaft sein.
- *FirewallResponse* (optional): Dieses neue Attribut beinhaltet das Ergebnis des Filterantwortverhaltens. Der Wert kann hierbei *keine Antwort*, *ICMP* oder *TCP RST* sein.
- *SequenceNumber* (optional): Dieses Attribut liefert die TCP Sequenznummer, die beim Sequenznummerntest ermittelt wurde.

Erste Phase

Die erste Phase des MFB Protokolls umfasst die Überprüfung, ob sich ein UPnP-fähiger NAT Router im lokalen Netzwerk befindet. Der MFB Client sendet hierfür eine UDP Multicast Nachricht an die Portnummer 1900. Befindet sich ein NAT Router mit UPnP Unterstützung im lokalen Netzwerk, antwortet dieser mit einer Nachricht, die einen Hyperlink zu einem XML Dokument beinhaltet. Dieses XML Dokument beinhaltet hierbei alle vom NAT Router unterstützten Dienste. Die Verfügbarkeit des Dienstes *AddPortMapping* zeigt hierbei an, ob eine Portzuordnung mittels UPnP erstellt werden kann.

Für eine ausführlichere Beschreibung des UPnP Protokolls sei an dieser Stelle auf Abschnitt 2.6.1 verwiesen.

Zweite Phase

Während der zweiten Phase des MFB Protokolls ermittelt der MFB Client das Filter- und das Filterantwortverhalten des verwendeten NAT Routers. Des Weiteren wird in dieser Phase überprüft, ob ein Raw Socket zur Verfügung steht und ob der

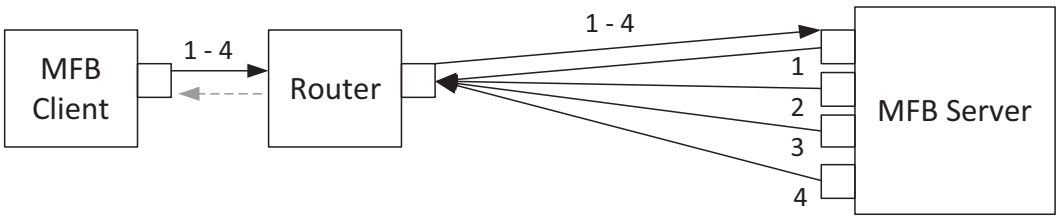


Abbildung 3.7: Diagramm zur Bestimmung des Filterverhaltens

NAT Router die TCP Sequenznummern ändert. Im letzten Teil der zweiten Phase wird ermittelt, wie der zur Verfügung stehende TCP Socket reagiert, wenn dieser geschlossen wird.

Der Client sendet vier *BindingRequest* Nachrichten jeweils per TCP und UDP parallel an den MFB Server. Zu diesem Zweck erstellt der MFB Client einen Socket für UDP und einen Socket für TCP, über die diese Nachrichten gesendet werden. Sollte dem MFB Client kein MFB Server bekannt sein, kann er dieselben Nachrichten an einen klassischen STUN Server senden. Aufgrund der Tatsache, dass ein klassischer STUN Server nur UDP unterstützt, ist der MFB Client nicht in der Lage eine TCP Verbindung zum klassischen STUN Server aufzubauen. Dementsprechend ist der Client nicht in der Lage die Netzwerkumgebung in Bezug auf TCP zu bestimmen.

Um das Filterverhalten des NAT Routers bestimmen zu können, beinhaltet jede *BindingRequest* Nachricht eine andere Kombination der Flags *ChangeIP* und *ChangePort*. Durch die verschiedenen Kombinationen der gesetzten Flags sendet der MFB Server die Antwortnachrichten jeweils über einen anderen IP Endpunkt (siehe Abbildung 3.7).

Der MFB Client wartet bis entweder alle Antworten empfangen wurden oder eine gegebene Zeitspanne abgelaufen ist. Abhängig davon, welche Antworten der NAT Router an den internen Rechner bzw. den MFB Client weiterleitet (gestrichelter Pfeil in Abbildung 3.7), kann der Client das Filterverhalten des NAT Routers bestimmen. In Tabelle 3.2 ist der Zusammenhang zwischen empfangenen Antworten und dem Filterverhalten dargestellt. Dieser Zusammenhang gilt für TCP und UDP.

Ein Unterschied in Bezug auf TCP ist allerdings, dass hierbei jeweils eine Verbindung aufgebaut werden muss, damit die Antwortnachricht gesendet bzw. empfangen werden kann. Die initiale Verbindung, die der MFB Client zum MFB Server aufgebaut hat, wird verwendet, um die vier *BindingRequest* Nachrichten zu senden und um die erste Antwortnachricht (*ChangeIP* und *ChangePort* nicht gesetzt) zu empfangen.

Empfangene Antworten	Filterverhalten
Antwort von *X:x* (1)	Adressen- und Port-abhängig
Antwort von *X:x* (1) und *X:y* (2)	Adressen-abhängig
Antwort von *X:x* (1) und *Y:x* (3)	Port-abhängig
Antwort von *X:x* (1), *X:y*, (2) *Y:x* (3), *Y:y* (4)	Endpunkt-unabhängig

Tabelle 3.2: Zusammenhang zwischen empfangenen Antworten und Filterverhalten

Um die verbliebenen drei Nachrichten zu senden, muss der MFB Server jeweils eine neue TCP Verbindung zum MFB Client aufbauen.

Die initiale TCP Verbindung wird zusätzlich noch verwendet, um den TCP Sequenznummerntest durchzuführen. Um den TCP Sequenznummerntest durchführen zu können, wird ein Raw Socket benötigt, der ausgehende TCP Pakete mitlesen kann. Aufgrund dessen, dass der Zugriff auf solch einen Raw Socket unter den meisten Betriebssystemen Administratorrechte benötigt, ist dieser Test möglicherweise nicht in allen Fällen durchführbar. Ist ein Raw Socket verfügbar, wird der TCP Sequenznummerntest durchgeführt. Der MFB Client setzt hierfür das *SequenceNumber_Request* Attribut und liest die TCP Sequenznummer des TCP Pakets aus mit dem die *BindingRequest* Nachricht gesendet wurde. Auf der anderen Seite liest der Server die TCP Sequenznummer der eingehenden TCP Nachrichten aus und setzt die ausgelesene TCP Sequenznummer als Wert des *SequenceNumber* Attributs in die Antwortnachricht. Durch einen Vergleich der beiden Sequenznummern kann der MFB Client feststellen, ob der NAT Router die TCP Sequenznummern ändert. Zusätzlich zum TCP Sequenznummerntest wird, wenn ein Raw Socket zur Verfügung steht, das Verhalten eines TCP Sockets überprüft, wenn dieser geschlossen wird. Hierfür initiiert der Client von einer beliebigen lokalen Portnummer aus einen TCP Verbindungsaufbau zum MFB Server. Anschließend wird der Socket direkt wieder geschlossen und mit Hilfe des Raw Socket überprüft, ob eine Fehlernachricht gesendet wurde.

Wenn der Client das Attribut *FirewallResponse_Request* in einer der *BindingRequest* Nachrichten gesetzt hat, führt der MFB Server den Test für das Filterantwortverhalten durch. Abhängig davon über welches Transportprotokoll die *BindingRequest* Nachricht empfangen wurde, sendet der MFB Server entweder einige UDP Pakete an zufällig gewählte externe Portnummern des MFB Clients oder versucht eine TCP Verbindung herzustellen. Der Server wartet anschließend eine bestimmte Zeit auf eine Antwort des NAT Routers. Wenn die ausgewählten Portnummern nicht zufäl-

Externer IP Endpunkt	Portzuordnungsverhalten
$MA(X{:}x) = MA(X{:}y) = MA(Y{:}y)$	Endpunkt-unabhängig
$MA(X{:}x) = MA(X{:}y) \neq MA(Y{:}y)$	Adressen-abhängig
$MA(X{:}x) \neq MA(X{:}y) = MA(Y{:}y)$	Port-abhängig
$MA(X{:}x) \neq MA(X{:}y) \neq MA(Y{:}y)$	Adressen- und Port-abhängig

Tabelle 3.3: Zusammenhang zwischen ermittelten externen IP Endpunkten und dem Portzuordnungsverhalten

lig zugeordnet sind und ein Endpunkt-unabhängiges Filterverhalten haben, kann der Server ermitteln, ob der NAT Router Pakete ohne Antwort verwirft, mit einer ICMP Nachricht oder einer TCP RST Nachricht antwortet. Das Ergebnis dieses Tests wird in die Antwortnachricht als Wert des Attributs *FirewallResponse* gesetzt.

Am Ende dieser Phase hat der Client Informationen über das Filterverhalten, das Filterantwortverhalten, die Verfügbarkeit eines Raw Sockets, das Verhalten eines TCP Sockets, wenn dieser geschlossen wird, sowie das Ergebnis des Sequenznummerntests. Zusätzlich zu diesen Informationen kennt der Client auch seinen externen IP Endpunkt *E:e*, der als Wert des *Mapped_Address* Attributes vom MFB Server gesetzt wurde.

Dritte Phase

Während der dritten Phase ermittelt der Client das Portzuordnungs- und das Portauswahlverhalten sowie die Unterstützung des NAT Routers für *Hairpinning*. Der Client sendet hierfür neue *BindingRequest* Nachrichten an die IP Endpunkte *X:x*, *X:y* und *Y:y* des MFB Servers. Um diese Nachrichten zu senden, werden dieselben Sockets mit denselben Portnummern verwendet wie in der zweiten Phase. Im Gegensatz zum Filtertest ist in keiner dieser Nachrichten eines der beiden Flags *ChangeIP* und *ChangePort* gesetzt. Der Server wird dementsprechend für alle Antwortnachrichten den IP Endpunkt verwenden über den die jeweilige *BindingRequest* Nachricht empfangen wurde.

Im Gegensatz zur zweiten Phase wird beim Test für das Portzuordnungsverhalten für jede gesendete *BindingRequest* Nachricht eine Antwort empfangen. Sobald der Client alle Antwortnachrichten empfangen hat, kann das Portzuordnungsverhalten bestimmt werden. Zu diesem Zweck werden die externen IP Endpunkte aus den *Mapped_Address* Attributen miteinander verglichen. In Tabelle 3.3 sind die

möglichen Werte der *Mapped_Address* Attribute und das dazugehörige Portzuordnungsverhalten dargestellt. Wenn beispielsweise alle IP Endpunkte der empfangenen *Mapped_Address* Attribute gleich sind, so hat der NAT Router ein Endpunkt-unabhängiges Portzuordnungsverhalten. Zusätzlich wird überprüft, ob die lokale Portnummer gleich der externen Portnummer ist. Wenn dies der Fall ist, so unterstützt der NAT Router die Eigenschaft der Porterhaltung. Dies gilt für UDP als auch für TCP.

Nachdem das Portzuordnungsverhalten bestimmt wurde, wird sofern das ermittelte Portzuordnungsverhalten kein Endpunkt-unabhängiges Verhalten ist, bestimmt, ob das Portauswahlverhalten vorhersagbar ist oder nicht. Hierfür wird überprüft, ob die vom NAT Router verwendeten externen Portnummern in einem kleinen Intervall (z.B. 10) liegen.

Parallel zur Bestimmung des Portzuordnungs- und Portauswahlverhaltens wird ermittelt, ob der NAT Router *Hairpinning* unterstützt. Zum diesem Zweck wird der externe IP Endpunkt benötigt, der in der zweiten Phase ermittelt wurde. Der Client versucht ein Paket an seinen eigenen externen IP Endpunkt *E:e* zu senden. Hierfür wird intern ein neuer Socket mit einer bisher nicht genutzten internen Portnummer verwendet. Der NAT Router unterstützt *Hairpinning*, wenn der Client in der Lage ist, dass selbst gesendete UDP Paket zu empfangen bzw. den TCP Verbindungsaufbau erfolgreich durchzuführen.

Nachdem alle Tests durchgeführt wurden, hat der Client am Ende dieser Phase das Portzuordnungs- und Portauswahlverhalten sowie die Unterstützung für die Porterhaltung und für *Hairpinning* ermittelt.

3.3.3 Peer-to-Peer MFB

In diesem Abschnitt wird beschrieben wie das MFB Protokoll, das in den vorherigen Abschnitten vorgestellt wurde, in einem Peer-to-Peer-basierten System eingesetzt werden kann. Das Ziel hierbei ist es, dass die Peers des Peer-to-Peer Netzwerks den MFB Dienst zur Verfügung stellen. Hierdurch kann die Notwendigkeit von dedizierten MFB Servern reduziert werden. Hierbei ergeben sich drei Herausforderungen, die im Folgenden näher betrachtet werden.

Die erste Herausforderung besteht darin, die öffentlichen IP Endpunkte der Peers, die den MFB Dienst zur Verfügung stellen (MFB Peers) zu ermitteln. Allerdings ist diese Herausforderung vergleichbar mit der Herausforderung einen aktiven Peer zu finden, um einem Peer-to-Peer Netzwerk beizutreten. Daher können bekannte

Bootstrapping-Verfahren (siehe beispielsweise [20, 45, 46]) verwendet werden, um die Adressen von aktiven MFB Peers zu ermitteln.

Die nächste Herausforderung besteht in der Tatsache, dass Peers in den meisten Fällen einen NAT Router verwenden und somit nicht direkt erreichbar sind. Eine Anforderung an einen MFB Peer ist, dass dieser auf zwei externen Portnummern erreichbar sein muss. Dies kann entweder mittels manueller Portweiterleitung, UPnP oder durch ein Endpunkt-unabhängiges Filterverhalten erreicht werden. Durch diese Anforderungen wird die Anzahl der möglichen MFB Peers reduziert. Allerdings werden Peers mit dieser Anforderungen unabhängig vom MFB Dienst in einem Peer-to-Peer Netzwerk benötigt (z.B. Bootstrapping oder für die Kommunikation mittels Mittelsmann), so dass davon ausgegangenen werden kann, das Peers mit diesen Eigenschaften existieren.

Die dritte und letzte Herausforderung liegt darin, dass Peers sehr selten mehr als eine öffentliche IP Adresse haben. In diesem Fall wäre ein MFB Peer nicht in der Lage eine Antwortnachricht auf eine *BindingRequest* Nachricht zu senden, in der das *ChangeIP* Flag gesetzt ist. Damit trotzdem eine Antwortnachricht an den anfragenden MFB Client gesendet werden kann, wird der Nachrichtendienst, der vom Peer-to-Peer Netzwerk bereitgestellt wird, verwendet. Der MFB Peer, der die Anfrage erhalten hat, setzt das *ChangeIP* Flag der Nachricht zurück und leitet diese Nachricht an einen Peer mit dem er direkt verbunden ist (z.B. ein Peer aus seiner Nachbarschaft). Dieser Peer kann dann eine Antwortnachricht erstellen und diese über seinen externen IP Endpunkt an den MFB Client zurück senden.

Dieser Ansatz bietet einige Vorteile. Der Peer, an den diese *BindingRequest* Nachricht weitergeleitet wird, muss nicht direkt erreichbar sein, um eine ausgehende *BindingResponse* Nachricht zu senden. Peers, die bereits im Peer-to-Peer Netzwerk integriert sind, können über den Peer-to-Peer Nachrichtendienst eine weitergeleitete *BindingRequest* Nachricht empfangen. Aufgrund dessen, dass die *BindingRequest* Nachricht an jeden beliebigen Peer weitergeleitet werden kann, muss ein MFB Peer nicht aktiv nach einem speziellen Peer suchen und die Verbindung mit ihm aufrechterhalten. Hierdurch können Verwaltungsaufwand und entsprechende Nachrichten eingespart werden.

Es existiert bei dem vorgestellten Ansatz eine mögliche Einschränkung bei der Bestimmung des Filterverhaltens. Wenn ein Peer, der eine weitergeleitete *BindingRequest* Nachricht empfängt in der nur das *ChangeIP* Flag gesetzt ist, sich hinter einem NAT Router befindet, der keine Porterhaltung unterstützt, kann nicht sichergestellt werden, dass beide Peers dieselbe externe Portnummer verwenden. Hierdurch

ist es möglich, dass der Client die Nachricht nicht empfängt oder fälschlicherweise ein Port-abhängiges Filterverhalten als ein Adressen- und Port-abhängiges Filterverhalten interpretiert. Um dies zu vermeiden, sollte ein MFB Peer die Nachricht nur an einen Peer weiterleiten, dessen lokale Portnummer auch als externe Portnummer verwendet wird (Porterhaltung). In Abschnitt 3.5 wird gezeigt, dass ein Port-abhängiges Filterverhalten nur eine sehr geringe praktische Relevanz hat, so dass die beschriebene Einschränkung in der Praxis vernachlässigt werden kann.

3.3.4 Auswahl des zu verwendenden NAT Traversal Verfahrens

Nachdem in den vorherigen Abschnitten das MFB Protokoll und die während der Ausführung des Protokolls ermittelten Eigenschaften eines NAT Routers beschrieben wurden, wird im folgenden Abschnitt beschrieben, wie diese Eigenschaften verwendet werden. Anhand der ermittelten Eigenschaften sowie der jeweiligen Bedingungen der verschiedenen NAT Traversal Verfahren (siehe Abschnitt 3.2.2) wird ein Regelwerk definiert, anhand dessen entschieden werden kann, welches Verfahren in einem konkreten Fall angewendet werden sollte. Zusätzlich zu der Entscheidung, welches Verfahren eingesetzt werden sollte, werden basierend auf den ermittelten Eigenschaften einige Verfahren parametrisiert, um die Erfolgswahrscheinlichkeit zu erhöhen. Welche Verfahren wie parametrisiert werden, wird im weiteren Verlauf dieses Abschnitts näher beschrieben.

Das Regelwerk, das in diesem Abschnitt vorgestellt wird, wird nicht alle Verfahren, die in dieser Arbeit betrachtet wurden, berücksichtigen. Warum die Verfahren Verbindungsumkehrung, NUTSS(a), P2PNAT und NATBLASTER nicht Bestandteil des Regelwerks sind, wird im Folgenden erläutert.

Das in Abschnitt 2.6.2 vorgestellte Verfahren der Verbindungsumkehrung ist kein eigenständiges Verfahren, sondern ein Spezialfall des direkten Verbindungsaufbaus. Die Verbindungsumkehrung ist ein direkter Verbindungsaufbau mit getauschten Rollen von A und B. Da dieser Rollentausch bei jedem Verfahren möglich ist, werden im Regelwerk nur die eigentlichen Verfahren betrachtet. Der Rollentausch wird für alle Verfahren gemeinsam betrachtet.

Das Verfahren NUTSS(a) wird für das Regelwerk nicht betrachtet, da dieses Verfahren auf IP-Spoofing basiert. Unter der Annahme, dass Internet Service Provider Ingress [26] und Egress Filter einsetzen, um Nachrichten, die mittels IP-Spoofing gesendet wurden, herauszufiltern, ist der Einsatz dieses Verfahrens in der Praxis deutlich eingeschränkt (siehe Abschnitt 2.2.3).

Eine Verbindung mit Hilfe des Verfahrens P2PNAT herzustellen setzt voraus, dass beide Kommunikationspartner den Verbindungsaufbau zur selben Zeit initiieren. Im Gegensatz zu den anderen Verfahren, ist bei diesem Verfahren die Zeitspanne für die Synchronisation geringer (siehe Abschnitt 3.2.2). In der Praxis ist die Einhaltung der Zeitspanne schwer zu gewährleisten, da man beispielsweise keinen Einfluss auf unterschiedliche Routing Pfade oder das Scheduling des Betriebssystems hat. Zusätzlich zu der Synchronisation hat P2PNAT höhere Anforderungen an die TCP Zustandsfilterung der involvierten NAT Router im Vergleich zu NUTSS(b), welches Bestandteil des Regelwerks ist. Aus diesen Gründen ist das Verfahren P2PNAT nicht Bestandteil des Regelwerks.

Das letzte Verfahren, das nicht Bestandteil des Regelwerks ist, ist NATBLASTER. Die Bedingungen, die erfüllt sein müssen, um mit dem Verfahren NATBLASTER erfolgreich eine Verbindung herzustellen, sind eine Obermenge der Bedingungen des Verfahrens SYNI. Daher würde das Verfahren nie ausgewählt und dementsprechend ist es kein Bestandteil des Regelwerks.

Im Folgenden werden die Regeln zur Auswahl, der für das Regelwerk relevanten Verfahren (direkter Verbindungsaufbau, NUTSS(b), SYNI sowie UDP Hole Punching) näher betrachtet. Bei diesen Regeln wird davon ausgegangen, dass A eine Verbindung zu B aufbauen möchte. A kann hierbei entscheiden, ob eine verlässliche Verbindung (TCP oder RUDP) oder ein unverlässlicher Nachrichtenaustausch per UDP gewünscht wird. Zu diesem Zweck besteht das Regelwerk aus zwei verschiedenen Teilen: ein Teil für TCP und ein Teil für UDP. Das Regelwerk für UDP wird auch angewendet, wenn versucht wird eine Verbindung per RUDP herzustellen. Abhängig von den ermittelten Eigenschaften, können die Rollen von A und B bei jedem Verfahren getauscht werden, wodurch die Richtung des Verbindungsaufbaus umgekehrt wird. Hierfür ist es allerdings notwendig, dass A in der Lage ist B über den Verbindungswunsch zu informieren, so dass B dann das ausgewählte Verfahren initiieren kann. Da die meisten Verfahren einen gemeinsamen Kommunikationskanal benötigten um Aktionen zu koordinieren oder Informationen auszutauschen, kann dieser Kanal auch verwendet werden, um einen umgekehrten Verbindungsaufbau zu initiieren.

Die aufgestellten Regeln werden in der Reihenfolge ausgewertet, in der sie im Regelwerk aufgelistet sind. Jede Regel wird mit den Standardrollen (A initiiert) sowie mit getauschten Rollen (B initiiert) ausgewertet. Die erste Regel, bei der alle Eigenschaften erfüllt sind, wird angewendet und somit das entsprechende Verfahren ausgeführt.

Sollte mit dem ausgewählten Verfahren keine Verbindung hergestellt werden können, wird das Regelwerk nach der zuletzt angewandten Regel weiter ausgewertet.

Regelwerk für TCP Im Folgenden wird das Regelwerk für TCP näher betrachtet. Das erste Verfahren, das für dieses Regelwerk betrachtet wird, ist der direkte Verbindungsaufbau. Bei diesem Verfahren muss B direkt erreichbar sein. Die Eigenschaft, dass B direkt erreichbar ist, ist erfüllt, wenn R_B ein Endpunkt-unabhängiges Filterverhalten hat oder wenn B durch die Verfügbarkeit von UPnP eine Portweiterleitung erstellen kann. Ist B mittels TCP direkt erreichbar, so kann A eine direkte Verbindung zu B mittels TCP aufbauen. Dieser Fall wird durch die folgenden Regeln beschrieben:

$$
\begin{aligned}
&1. EndpunktUnabhaengigesFilterverhalten(B, TCP) \rightarrow Direkt(A, B, TCP) \\
&2. UPnPVerfuegbar(B, TCP) \rightarrow PortweiterleitungErstellen(B, TCP) \wedge \\
&\quad Direkt(A, B, TCP)
\end{aligned}
\tag{3.1}
$$

Konnte keine der beiden Regeln angewendet werden, so kann unter weiteren Bedingungen eine Verbindung mit dem direkten Verbindungsaufbau hergestellt werden. Hierfür muss R_B ein Port-abhängiges Filterverhalten aufweisen. Das bedeutet, dass alle Pakete, die von einer bestimmten externen Portnummer aus gesendet werden, vom Router weitergeleitet werden. Damit mit diesem Filterverhalten ein erfolgreicher Verbindungsaufbau möglich ist, muss A die verwendete externe Portnummer von R_A beeinflussen können. Damit A die verwendete externe Portnummer beeinflussen kann, muss R_A die Eigenschaft der Porterhaltung haben. Zusätzlich muss die benötigte Portnummer auf der Seite von A verfügbar sein, das bedeutet, dass die Portnummer noch nicht zugeordnet sein darf. Wenn R_B ein Port-abhängiges Filterverhalten hat und beispielsweise alle Pakete, die von der externen Portnummer 4000 gesendet werden an den internen Rechner B weiterleitet, muss R_A die externe Portnummer 4000 verwenden um Pakete an R_B senden zu können. Verwendet A den lokalen Port 4000 und R_A unterstützt die Porterhaltung und zusätzlich ist die externe Portnummer 4000 noch nicht zugeordnet, dann wird R_A die externe Portnummer 4000 verwenden und es kann erfolgreich eine Verbindung aufgebaut werden. Dieser Verbindungsaufbau wird durch die folgende Regel 3.2 beschrieben.

$$3.PortAbhaengigesFilterverhalten(B,TCP) \wedge Porterhaltung(A,TCP) \wedge PortVerfuegbar(A,TCP) \rightarrow Direkt(A,B,TCP) \tag{3.2}$$

Sollte es nicht möglich sein mittels direkten Verbindungsaufbaus eine TCP Verbindung herzustellen, muss eines der in Abschnitt 3.2 vorgestellten TCP Hole Punching Verfahren eingesetzt werden. Alle TCP Hole Punching Verfahren setzen voraus, dass der externe IP Endpunkt, der zur Kommunikation verwendet wird, verlässlich bestimmt werden kann. Hierfür muss der NAT Router ein Endpunkt-unabhängiges Portzuordnungsverhalten haben oder das Portauswahlverhalten muss einem erkennbaren Muster folgen und so entsprechend vorhersagbar sein.

Sind die externen IP Endpunkte der beiden Kommunikationspartner bestimmbar, ist die erste Bedingung für die beiden TCP Hole Punching Verfahren NUTSS(b) und SYNI erfüllt. An dieser Stelle des Regelwerks muss nun entschieden werden, welches dieser beiden Hole Punching Verfahren in einer bestimmten Situation verwendet werden sollte. Beide Verfahren haben unterschiedliche Anforderungen an die TCP Zustandsfilterung, der beim Verbindungsaufbau involvierten NAT Router. Allerdings wird das Verhalten der TCP Zustandsfilterung nicht durch das MFB Protokoll bestimmt (siehe Abschnitt 3.3.1). Ohne die Informationen über die Zustandsfilterung ist es allerdings nicht möglich eine verlässliche Aussage darüber zu treffen mit welchem Verfahren eine Verbindung hergestellt werden kann. An dieser Stelle wird daher die Reihenfolge der Regeln in Abhängigkeit der allgemeinen Erfolgswahrscheinlichkeit festgelegt. In Abschnitt 3.5.1 wird gezeigt, dass das Verfahren SYNI im Allgemeinen die höchste Erfolgswahrscheinlichkeit aufweist. Infolge dessen wird erst überprüft, ob das Verfahren SYNI angewendet werden kann und anschließend werden die Eigenschaften für NUTSS(b) überprüft.

$$4.ExternerIPEPBestimmbar(A,TCP) \wedge RawSocket(A) \wedge NoTCPSeqChange(A) \wedge ExternerIPEPBestimmbar(B,TCP) \wedge RawSocket(B) \wedge SilentDrop(B,TCP) \rightarrow SYNI_{StandardTTL}(A,B) \tag{3.3}$$

$$5.ExternerIPEPBestimmbar(A,TCP) \wedge RawSocket(A) \wedge NoTCPSeqChange(A) \wedge ExternerIPEPBestimmbar(B,TCP) \wedge RawSocket(B) \rightarrow SYNI(A,B)$$

In Abschnitt 3.2.1 wurde beschrieben, dass beim Verfahren SYNI die TTL während des Verbindungsaufbaus auf der Seite von A geändert werden muss. Wenn R_B ein Filterantwortverhalten vom Typ *ohne Antwort verwerfen* (SilentDrop) hat, ist die TTL Modifikation allerdings nicht notwendig, da das ausgehende SYN mit Standard TTL keine Auswirkungen hat. In diesem Fall kann das SYNI Verfahren in einer modifizierten Variante mit Standard TTL ausgeführt werden (siehe Regel 3.3).

Sollte das Verfahren SYNI aufgrund der durch das MFB Protokoll ermittelten Eigenschaften nicht eingesetzt werden können, wird als nächstes überprüft, ob das Verfahren NUTSS(b) eingesetzt werden kann. Voraussetzung für NUTSS(b) ist, dass der verwendete Socket kein TCP RST sendet, wenn dieser geschlossen wird. Äquivalent zum Verfahren SYNI kann NUTSS(b) das erste SYN Paket mit einem Standard TTL Wert senden, wenn R_B auf dieses Paket nicht mit einer Fehlernachricht reagiert (siehe Regel 3.4).

$$\begin{aligned} 6.&\mathit{ExternerIPEPBestimmbar}(A,TCP) \wedge \mathit{SocketSchliessenKeinRST}(A) \wedge \\ &\mathit{ExternerIPEPBestimmbar}(B,TCP) \wedge \mathit{SilentDrop}(B,TCP) \\ &\rightarrow NUTSS_{StandardTTL}(A,B) \end{aligned} \tag{3.4}$$

$$7.\mathit{ExternerIPEPBestimmbar}(A,TCP) \wedge \mathit{SocketSchliessenKeinRST}(A) \wedge \mathit{ExternerIPEPBestimmbar}(B,TCP) \rightarrow NUTSS(A,B)$$

Sollte der Socket beim Schließen ein TCP RST senden, so kann auch NUTSS(b) in einer weiteren modifizierten Variante ausgeführt werden, wenn A einen Raw Socket zur Verfügung hat. In diesem Fall würde mit Hilfe des Raw Sockets das erste SYN Paket gesendet, so dass hierfür kein Socket geöffnet und anschließend wieder geschlossen werden muss.

$$\begin{aligned} 8.&\mathit{ExternerIPEPBestimmbar}(A,TCP) \wedge \mathit{RawSocket}(A) \wedge \\ &\mathit{ExternerIPEPBestimmbar}(B,TCP) \wedge \mathit{SilentDrop}(B,TCP) \\ &\rightarrow NUTSS_{RawStandardTTL}(A,B) \end{aligned} \tag{3.5}$$

$$9.\mathit{ExternerIPEPBestimmbar}(A,TCP) \wedge \mathit{RawSocket}(A) \wedge \mathit{ExternerIPEPBestimmbar}(B,TCP) \rightarrow NUTSS_{Raw}(A,B)$$

Diese Variante ist in Regel 3.5 darstellt und lässt sich ebenso wie die ursprüngliche Version in einer Variante mit Standard TTL verwenden.

Im Folgenden ist das gesamte Regelwerk für TCP (siehe Regel 3.6) zusammenfassend dargestellt.

1. $EndpunktUnabhaengigesFilterverhalten(B, TCP) \rightarrow Direkt(A, B, TCP)$

2. $UPnPVerfuegbar(B, TCP) \rightarrow PortweiterleitungErstellen(B, TCP) \wedge$
   $Direkt(A, B, TCP)$

3. $PortAbhaengigesFilterverhalten(B, TCP) \wedge Porterhaltung(A, TCP) \wedge$
   $PortVerfuegbar(A, TCP) \rightarrow Direkt(A, B, TCP)$

4. $ExternerIPEPBestimmbar(A, TCP) \wedge RawSocket(A) \wedge$
   $NoTCPSeqChange(A) \wedge ExternerIPEPBestimmbar(B, TCP) \wedge$
   $RawSocket(B) \wedge SilentDrop(B, TCP) \rightarrow SYNI_{StandardTTL}(A, B)$

5. $ExternerIPEPBestimmbar(A, TCP) \wedge RawSocket(A) \wedge$
   $NoTCPSeqChange(A) \wedge ExternerIPEPBestimmbar(B, TCP) \wedge$
   $RawSocket(B) \rightarrow SYNI(A, B)$

6. $ExternerIPEPBestimmbar(A, TCP) \wedge SocketSchliessenKeinRST(A) \wedge$
   $ExternerIPEPBestimmbar(B, TCP) \wedge SilentDrop(B, TCP)$
   $\rightarrow NUTSS_{StandardTTL}(A, B)$

7. $ExternerIPEPBestimmbar(A, TCP) \wedge SocketSchliessenKeinRST(A) \wedge$
   $ExternerIPEPBestimmbar(B, TCP) \rightarrow NUTSS(A, B)$

8. $ExternerIPEPBestimmbar(A, TCP) \wedge RawSocket(A) \wedge$
   $ExternerIPEPBestimmbar(B, TCP) \wedge SilentDrop(B, TCP)$
   $\rightarrow NUTSS_{RawStandardTTL}(A, B)$

$$9.ExternerIPEPBestimmbar(A,TCP) \wedge RawSocket(A) \wedge \\ ExternerIPEPBestimmbar(B,TCP) \rightarrow NUTSS_{Raw}(A,B) \tag{3.6}$$

Regelwerk für UDP Nachdem alle Regeln zur Auswahl eines TCP basierten Verfahrens betrachtet wurden, wird im Folgenden das Regelwerk für UDP näher betrachtet. Dieses Regelwerk wird verwendet, wenn explizit ein UDP Nachrichtenaustausch gewünscht wird oder wenn versucht wird eine RUDP Verbindung als Alternative zu einer TCP Verbindung aufzubauen.

Analog zum Regelwerk für TCP wird als erstes überprüft, ob ein direkter Verbindungsaufbau möglich ist. Sollte B direkt per UDP erreichbar sein, so kann A B direkt per UDP kontaktieren (siehe Regel 3.7). Auch hier bedeutet direkt erreichbar, dass R_B ein Endpunkt-unabhängiges Filterverhalten hat oder B durch die Verfügbarkeit von UPnP eine Portweiterleitung erstellen kann.

$$1.EndpunktUnabhaengigesFilterverhalten(B,UDP) \rightarrow Direkt(A,B,UDP)$$

$$2.UPnP(B,UDP) \rightarrow PortweiterleitungErstellen(B,UDP) \wedge Direkt(A,B,UDP) \tag{3.7}$$

Ist B nicht direkt per UDP erreichbar, so kann äquivalent zum direkten Verbindungsaufbau bei TCP ein direkter Nachrichtenaustausch per UDP hergestellt werden, wenn R_B ein Port-abhängiges Filterverhalten hat und R_A die Porterhaltung unterstützt (siehe Regel 3.8).

$$3.PortAbhaengigesFilterverhalten(B,UDP) \wedge Porterhaltung(A,UDP) \wedge \\ PortVerfuegbar(A,UDP) \rightarrow Direkt(A,B,UDP) \tag{3.8}$$

Für den Fall, dass keine der vorherigen Regeln für UDP angewendet werden konnte, wird als nächstes geprüft, ob das Verfahren UDP Hole Punching angewendet werden kann. Hierbei müssen auf beiden Seiten die externen IP Endpunkte bestimmbar sein. Analog zu den Verfahren SYNI und NUTSS(b) kann dieses Verfahren mit einem geringen TTL Wert als auch mit dem Standard TTL Wert ausgeführt werden (siehe Regel 3.9).

$$
\begin{aligned}
&4.ExternerIPEPBestimmbar(A,UDP) \wedge SilentDrop(B,UDP) \wedge \\
&\quad ExternerIPEPBestimmbar(B,UDP) \\
&\quad \rightarrow UDPHolePunching_{StandardTTL}(A,B) \\
&5.ExternerIPEPBestimmbar(A,UDP) \wedge ExternerIPEPBestimmbar(B,UDP) \\
&\quad \rightarrow UDPHolePunching(A,B)
\end{aligned}
\tag{3.9}
$$

Das vollständige Regelwerk für UDP ist in Regel 3.10 zusammengefasst.

$$
\begin{aligned}
&1.EndpunktUnabhaengigesFilterverhalten(B,UDP) \rightarrow Direkt(A,B,UDP) \\
&2.UPnP(B,UDP) \rightarrow PortweiterleitungErstellen(B,UDP) \wedge Direkt(A,B,UDP) \\
&3.PortAbhaengigesFilterverhalten(B,UDP) \wedge Porterhaltung(A,UDP) \wedge \\
&\quad PortVerfuegbar(A,UDP) \rightarrow Direkt(A,B,UDP) \\
&4.ExternerIPEPBestimmbar(A,UDP) \wedge SilentDrop(B,UDP) \wedge \\
&\quad ExternerIPEPBestimmbar(B,UDP) \\
&\quad \rightarrow UDPHolePunching_{StandardTTL}(A,B) \\
&5.ExternerIPEPBestimmbar(A,UDP) \wedge ExternerIPEPBestimmbar(B,UDP) \\
&\quad \rightarrow UDPHolePunching(A,B)
\end{aligned}
\tag{3.10}
$$

3.3.5 Verwandte Arbeiten

Nachdem im vorherigen Abschnitt das MFB Protokoll, dessen Adaption für Peer-to-Peer Netzwerke sowie das Regelwerk zur Auswahl eines NAT Traversal Verfahrens beschrieben wurden, werden im Folgenden verwandte Arbeiten in diesem Bereich näher betrachtet. Die vorgestellten Arbeiten werden über die in Abschnitt 3.3.1 definierten Anforderungen mit dem MFB Protokoll verglichen.

Wie in Abschnitt 2.5 beschrieben, handelt es sich bei dem klassischen STUN Protokoll um ein simples auf UDP basierendes Protokoll, um das Vorhandensein und Verhalten eines NAT Routers zu bestimmen. Im Gegensatz zum MFB Protokoll verwendet das klassische STUN Protokoll eine eigene Klassifikation und kann damit nur zwischen vier NAT Typen *Full Cone*, *Restricted Cone*, *Port Restricted Cone* und *Symmetric* NAT unterscheiden. Diese Klassifikation ist zu restriktiv, so dass nicht alle Router exakt mit dieser Klassifikation beschrieben werden können [68]. Diese Restriktion resultiert daraus, dass nur vier Typen verwendet werden, wobei das Portzuordnungs- und Filterverhalten, wie in [68] und Abschnitt 2.4 beschrieben, 16 verschiedene Verhaltensweisen unterscheidet. In Tabelle 2.3 sind die klassischen STUN Typen und die entsprechenden Kombinationen aus Portzuordnungs- und Filterverhalten dargestellt. Ein weiterer Unterschied ist, dass das klassische STUN Protokoll vollständig auf UDP basiert. Daher ist das klassische STUN Protokoll nicht in der Lage Informationen über das Verhalten beim Einsatz von TCP zu bestimmen. In Bezug auf die definierten Anforderungen, erfüllt das klassische STUN Protokoll keine der Anforderungen. Basierend auf dem klassischen STUN Protokoll existieren weitere zahlreiche Veröffentlichungen [57, 58, 70] zur Ermittlung von NAT Verhaltensweisen oder im Bereich NAT Traversal, welche allerdings alle auf der klassischen STUN Klassifikation basieren. Daher können alle diese Ansätze die Anforderungen A2, A3 und A10 nicht erfüllen.

Eine weitere Arbeit in diesem Gebiet ist das auf dem klassischen STUN Protokoll basierende STUNT Protokoll [30]. STUNT erweitert hierbei das klassische STUN Protokoll um die Unterstützung für TCP, wodurch die Anforderung A10 erfüllt wird. Allerdings werden auch von STUNT die Anforderungen A2 und A3 nicht erfüllt.

Das klassische STUN Protokoll wurde in einer nachfolgenden Arbeit [68] grundlegend überarbeitet. Die neue Spezifikation ersetzt die NAT Klassifikation des klassischen STUN Protokolls mit der Kombination aus Portzuordnungs- und Filterverhalten [68]. Zusätzlich wurde der Algorithmus zur Bestimmung des NAT Verhaltens (siehe Abbildung 2.9) dahingehend reduziert, dass nur noch der externe IP Endpunkt des Clients bestimmt werden kann. Das MFB Protokoll und das STUN Protokoll verwenden somit die gleiche Klassifikation, allerdings kann das STUN Protokoll das Verhalten gemäß der verwendeten Spezifikation nicht bestimmen. Daher erfüllt das STUN Protokoll keine der aufgestellten Anforderungen.

MacDonald und Lowekamp haben in [52] ein Protokoll mit dem Namen *NAT Behavior Discovery Using STUN* (NAT B.D.) vorgestellt. Hierbei haben die Autoren das klassische STUN Protokoll so modifiziert, dass es das Verhalten eines NAT Routers

Ansätze	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11
STUN (klassisch)	-	-	-	-	-	-	-	-	-	-	-
STUNT	-	-	-	-	-	-	-	-	-	✓	-
STUN	-	-	-	-	-	-	-	-	-	-	-
NAT B.D.	-	✓	✓	-	-	-	-	-	-	✓	✓
MFB Protokoll	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

A1 Verfügbarkeit von UPnP
A2 Filterverhalten
A3 Portzuordnungsverhalten
A4 Porterhaltung
A5 Portauswahlverhalten
A6 Filterantwortverhalten
A7 TCP Sequenznummern Modifikation
A8 Verfügbarkeit von Raw Sockets
A9 TCP RST nachdem der Socket geschlossen wurde
A10 Separate Ergebnisse für TCP und UDP
A11 Unterstützung von Hairpinning

Tabelle 3.4: Vergleich des MFB Protokolls mit verwandten Arbeiten

gemäß des Portzuordnungs- und Filterverhaltens [68] bestimmt. Zusätzlich wurde das Protokoll um die Unterstützung für TCP und die Bestimmung von *Hairpinning* erweitert. Hierfür verwendet das Protokoll andere Attribute als das klassische STUN Protokoll und zusätzlich wurden neue Attribute eingeführt, die als gefordert definiert sind. Der Nachteil, der sich hieraus ergibt ist, dass das NAT B.D. Protokoll inkompatibel zum klassischen STUN Protokoll ist, wodurch existierende klassische STUN Server nicht verwendet werden können. Des Weiteren erfüllt das Protokoll die Anforderungen A1 und A4 bis A9 nicht.

In Tabelle 3.4 sind abschließend alle verwandten Arbeiten sowie die Anforderungen, wie sie in Abschnitt 3.3.1 definiert wurden, zusammengefasst.

3.4 Testumgebung

Um die in den vorherigen Abschnitten vorgestellten Protokolle und Verfahren zu evaluieren, wurde im Rahmen dieser Arbeit eine spezielle Testumgebung entwickelt.

Diese Testumgebung besteht zum einem aus einem Hardwareaufbau und einer Softwarekomponente, die in den nächsten Abschnitten genauer betrachtet werden.

3.4.1 Hardware

Der Hardwareaufbau besteht aus zwei Kommunikationspartnern, zwei Servern und einer Menge von NAT Routern, wie in Abbildung 3.8 dargestellt. Jeder NAT Router verfügt über eine Schnittstelle für das *Wide Area Network* (WAN) und mindestens eine Schnittstelle für das lokale Netzwerk (LAN). Das Protokoll, das von den NAT Routern auf der Seite des WAN Anschlusses verwendet wird, ist das Point-to-Point Protokoll (PPP), welches in Verbindung mit Ethernet verwendet wird (PPPoE). Alle Router sind über den WAN Anschluss mit einem Switch und darüber mit einem PPPoE Server verbunden. Jeder NAT Router registriert sich automatisch per PPPoE beim Server.

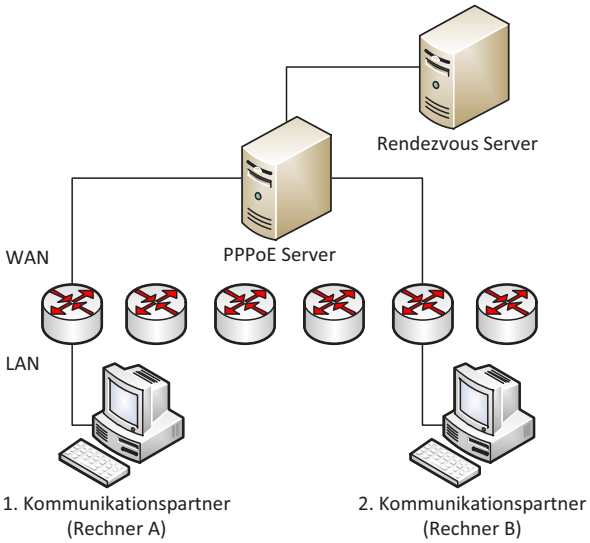


Abbildung 3.8: Allgemeiner Hardwareaufbau der Testumgebung

Der PPPoE Server weist den NAT Routern, wenn sie sich registrieren, eine IP Adresse zu, die vom NAT Router als *externe IP Adresse* verwendet wird. Zusätzlich dient der PPPoE Server dazu, Pakete zwischen den einzelnen NAT Routern weiterzuleiten und den Netzwerkverkehr zwischen diesen zu speichern. Der PPPoE Server sowie die verwendeten NAT Router befinden sich im selben externen Netzwerk (172.30.1.0). Zusätzlich befindet sich im externen Netzwerk noch ein *Rendezvous Server*. Dieser Server stellt den MFB Dienst zur Verfügung, so dass Rechner A und Rechner B das Verhalten der verwendeten NAT Router bestimmen können.

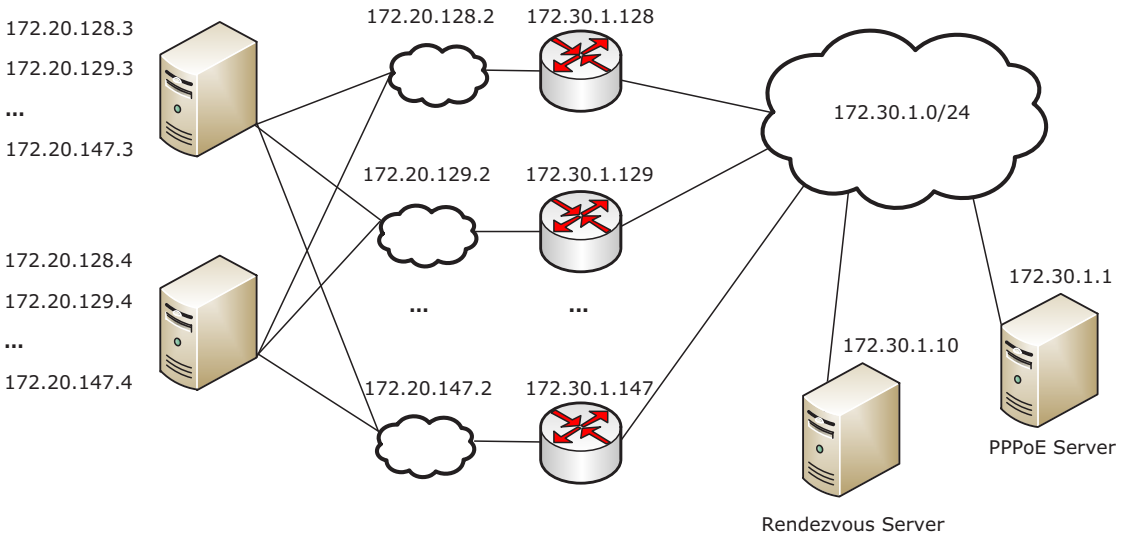


Abbildung 3.9: Netzwerkaufbau der Testumgebung

Jeder NAT Router verfügt zusätzlich zu der externen IP Adresse aus dem externen Netzwerk über eine IP Adresse aus einem privaten Netzwerk, das über die LAN Schnittstelle erreichbar ist. Die beiden Kommunikationspartner verfügen jeweils über eine IP Adresse aus jedem der privaten Netzwerke (siehe Abbildung 3.9). Hierdurch sind die Kommunikationspartner in der Lage mit jedem NAT Router über dessen LAN Anschluss zu kommunizieren. In Abbildung 3.9 ist die Netzwerkstruktur des gesamten Testsystems grafisch dargestellt.

3.4.2 Software

Basierend auf dem im vorherigen Abschnitt beschriebenen Hardwareaufbau wurde eine Software entwickelt mit der NAT Traversal Verfahren automatisch mit verschiedenen NAT Routern ausgeführt werden können. Im Folgenden wird eine Reihe von Anforderungen definiert, die von der Software erfüllt werden muss. Anschließend werden die Schnittstellen für die Implementierung der NAT Traversal Verfahren sowie der allgemeine Programmablauf beschrieben.

Automatische Nutzung der ausgewählten NAT Router Die Software soll in der Lage sein ein NAT Traversal Verfahren mit einer beliebigen Kombination von NAT Routern durchzuführen. Nachdem eine Kombination von NAT Routern ausgewählt wurde, muss sicherstellt werden, dass die beiden Kommunikationspartner jeweils einen der ausgewählten NAT Router für die Kommunikation verwenden.

Ausführung verschiedener NAT Traversal Verfahren Zusätzlich zur Auswahl einer beliebigen Kombination an NAT Routern, soll die Software in der Lage sein jedes implementierte NAT Traversal Verfahren automatisch durchzuführen. Neue Verfahren sollen hierbei einfach integrierbar sein.

Bestimmung des externen IP Endpunkts Die meisten NAT Traversal Verfahren benötigen den externen IP Endpunkt von einem oder beiden Kommunikationspartnern. Daher soll die Software in der Lage sein, den externen IP Endpunkt für den ausgewählten NAT Router zu bestimmen und diesen dem ausgewählten NAT Traversal Verfahren zur Verfügung zu stellen.

Gemeinsamer Kommunikationskanal Zusätzlich zu der Bestimmung des externen IP Endpunkts benötigen NAT Traversal Verfahren häufig einen gemeinsamen Kommunikationskanal. Damit dieser Kommunikationskanal nicht für jedes Verfahren neu entwickelt werden muss, muss die Software den NAT Traversal Verfahren einen gemeinsamen Kommunikationskanal zur Verfügung stellen.

Netzwerkmitschnitt Um eine aussagekräftige und verlässliche Evaluation der einzelnen NAT Traversal Verfahren zu erhalten, wird ein Netzwerkmitschnitt benötigt. Dieser Netzwerkmitschnitt muss die Kommunikation zwischen Kommunikationspartner A und dem verwendeten NAT Router R_A, zwischen R_A und dem vom Kommunikationspartner B verwendeten NAT Router R_B sowie die Kommunikation zwischen R_B und B beinhalten. Hierdurch erst ist es möglich eine exakte Aussage darüber zu treffen, warum ein Verfahren funktioniert hat oder warum nicht.

Schnittstellen

Nachdem im vorherigen Abschnitt die Anforderungen an die Software definiert wurden, wird im folgenden Abschnitt beschrieben, welche Schnittstelle die NAT Traversal Verfahren implementieren müssen, damit diese mit der Software ausgeführt werden können.

In der Auflistung 3.1 ist die allgemeine Schnittstelle beschrieben, die jedes NAT Traversal Verfahren implementieren muss, damit es in der Testumgebung ausgeführt werden kann. Die Implementierung eines Verfahrens besteht aus zwei verschiedenen Teilen. Der erste Teil umfasst die Funktionalität für den Kommunikationspartner A und der zweite Teil die für den Kommunikationspartner B. Jeder Teil muss hierbei

```
1  public interface INATTraversal
2  {
3      void StartHost(IPEndPoint localInternalIPEP, AsyncCallback<
           ReturnInformationEventArgs> callback);
4  }
```

Auflistung 3.1: Allgemeine Schnittstelle für NAT Traversal Verfahren

die allgemeine Schnittstelle implementieren. Die Methode *StartHost* startet hierbei asynchron das NAT Traversal Verfahren auf einem Rechner. Als Parameter werden der Methode der lokale IP Endpunkt sowie ein *Callback* übergeben. Der lokale IP Endpunkt besteht zum einen aus einer IP Adresse, die aus dem privaten Netzwerk des zu verwendenden NAT Routers stammt sowie einer ausgewählten lokalen Portnummer. Mit Hilfe des Callbacks wird signalisiert, wann das Verfahren beendet ist und ob eine Verbindung aufgebaut werden konnte oder nicht.

```
1  public interface INATTraversalWithOoBCommunication
2  {
3      event EventHandler<SendMessageEventArgs> SendOoBMessage;
4
5      void HandleOoBMessage(byte[] msg);
6  }
```

Auflistung 3.2: Schnittstelle für die Nutzung eines gemeinsamen Kommunikationskanals

Einige der vorgestellten NAT Traversal Verfahren benötigen einen gemeinsamen Kommunikationskanal. Verfahren, die dies benötigen, müssen zusätzlich die Schnittstelle, die in Auflistung 3.2 dargestellt ist, implementieren. Mit Hilfe des Ereignisses *SendOoBMessage* kann das NAT Traversal Verfahren eine Nachricht an das aktuell ausgeführte Verfahren des anderen Kommunikationspartners senden. Wenn die Nachricht beim Kommunikationspartner angekommen ist, wird die Nachricht über die Methode *HandleOoBMessage* an das aktuell ausgeführte Verfahren übergeben.

```
1  public interface INATTraversalWithRemoteExternalIPEPRequired
2  {
3      IPEndPoint RemoteExternalIPEP { get; set; }
4  }
```

Auflistung 3.3: Schnittstelle für den externen IP Endpunkt

Zusätzlich müssen einige NAT Traversal Verfahren die externen IP Endpunkte bestimmen und austauschen. Wenn ein Verfahren dies benötigt, muss die in Auflis-

tung 3.3 beschriebene Schnittstelle implementiert werden. Implementiert ein Verfahren diese Schnittstelle, so wird der externe IP Endpunkt des Kommunikationspartners von der Software ermittelt und an das NAT Traversal Verfahren übergeben.

```
1  public interface INATTraversalWithRawSocket
2  {
3      WinPcapDevice RawSocket { get; set; }
4  }
```

Auflistung 3.4: Schnittstelle für Raw Sockets

NAT Traversal Verfahren wie beispielsweise NATBLASTER oder SYNI benötigen einen speziellen Raw Socket um eigene TCP Pakete zu senden. Implementiert ein NAT Traversal Verfahren die Schnittstelle aus Auflistung 3.4, wird ein solcher Raw Socket über die Eigenschaft *RawSocket* an das NAT Traversal Verfahren übergeben.

```
1  public interface INATTraversalLowTTLSupported
2  {
3      bool UseLowTTL { get; set; }
4  }
```

Auflistung 3.5: Schnittstelle zur Verwendung eines geringen TTL Wertes

Einige NAT Traversal Verfahren können in einer Version mit einem Standard TTL Wert oder einem geringen TTL Wert ausgeführt werden. Wenn ein Verfahren beide Möglichkeiten bietet, muss die in Auflistung 3.5 dargestellte Schnittstelle implementiert werden. Mit Hilfe der Eigenschaft *UseLowTTL* kann dem NAT Traversal Verfahren signalisiert werden einen geringen TTL Wert anstatt des Standard TTL Wertes zu verwenden.

Implementierung

Im folgenden Abschnitt wird die Implementierung der Softwarekomponente näher erläutert und beschrieben, wie die in Abschnitt 3.4.2 aufgestellten Anforderungen erfüllt werden. Die Software wurde in der Programmiersprache .NET C# implementiert und verwendet das Framework WinPcap [17]. Dieses Framework stellt einen Raw Socket zur Verfügung mit dem eigene TCP Pakete gesendet werden können.

Die Software besteht aus zwei zentralen Komponenten. Die erste Komponente ist der *Master*, die auf Rechner A ausgeführt wird und die zweite Komponente ist der

Slave, die auf Rechner B ausgeführt wird. Der Master koordiniert den gesamten Ablauf und ist zuständig für die Auswahl der NAT Router und des auszuführenden NAT Traversal Verfahrens. Alle NAT Traversal Verfahren, die mit Hilfe der Software ausgeführt werden sollen, müssen hierbei die im vorherigen Abschnitt beschriebene allgemeine Schnittstelle implementieren.

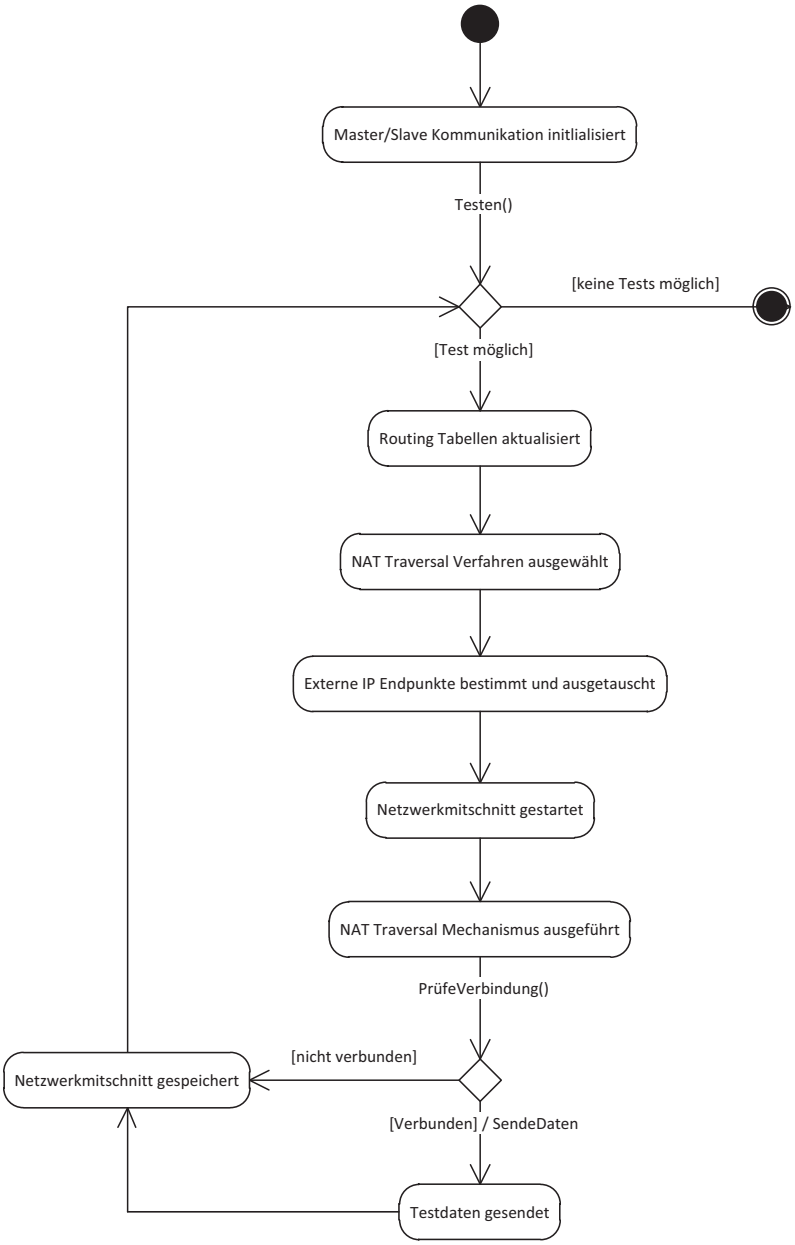


Abbildung 3.10: Ablaufdiagramm der Software

In Abbildung 3.10 ist der Programmlauf aus Sicht des Masters dargestellt. Im ersten Schritt wird die Kommunikation zwischen dem Master und dem Slave hergestellt. Anschließend wählt der Master die beiden NAT Router aus, die für den Testdurch-

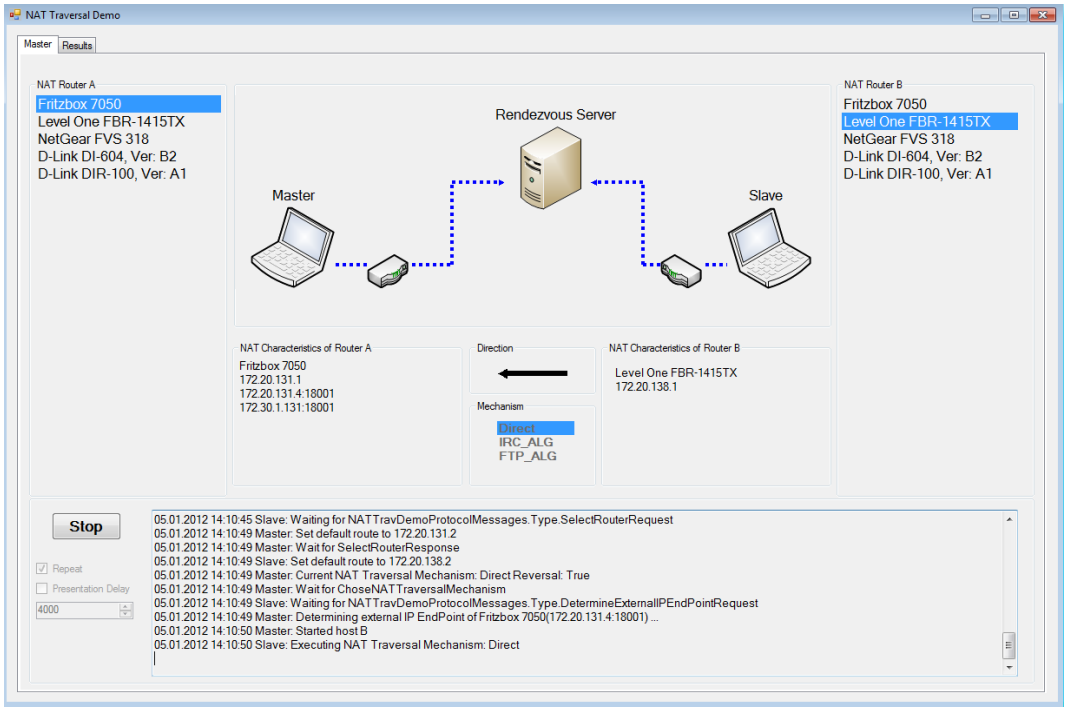


Abbildung 3.11: Screenshot der Software der Testumgebung

gang verwendet werden sollen. Zu diesem Zweck sendet der Master eine Nachricht an den Slave. Anschließend werden vom Master und Slave entsprechende Einträge in den Routing Tabellen der Betriebssysteme erstellt, so dass alle Pakete, die für den Testdurchgang gesendet werden, immer an die ausgewählten NAT Router gesendet werden.

Der nächste Schritt besteht darin, das NAT Traversal Verfahren auszuwählen und wenn benötigt, die externen IP Endpunkte zu bestimmen und auszutauschen. Bevor das eigentliche NAT Traversal Verfahren ausgeführt wird, müssen der Master, der Slave sowie der PPPoE Server den Netzwerkmitschnitt starten. Im Anschluss an diesen Schritt wird das ausgewählte Verfahren gestartet.

Nachdem beide Kommunikationspartner das NAT Traversal Verfahren gestartet haben, wartet die Testsoftware darauf, dass das Verfahren terminiert. Konnte mit dem ausgewählten NAT Traversal Verfahren eine Verbindung hergestellt werden, werden von den beiden Kommunikationspartnern Daten über diese Verbindung ausgetauscht. Unabhängig davon, ob eine Verbindung aufgebaut werden konnte oder nicht, werden nach dem Durchlauf, die Netzwerkmitschnitte vom PPPoE Server sowie der vom Master und Slave zusammengefasst und gespeichert.

Abbildung 3.12: Foto des Laboraufbaus der Testumgebung

Abschließend prüft der Master, ob weitere Kombinationen von NAT Routern getestet oder weitere NAT Traversal Verfahren durchgeführt werden sollen und startet gegebenenfalls einen weiteren Durchlauf.

In Abbildung 3.11 ist ein Screenshot der Software dargestellt und Abbildung 3.12 zeigt ein Foto des Laboraufbaus der gesamten Testumgebung.

3.5 Evaluation

Im folgenden Abschnitt werden die Verfahren SYNI sowie das MFB Protokoll evaluiert. Das Verfahren SYNI wird hinsichtlich seiner Erfolgswahrscheinlichkeit unter Verwendung verschiedener NAT Router evaluiert. Die Evaluation des MFB Protokolls umfasst die Bestimmung der Verhaltensweisen verschiedener NAT Router sowie die Bestimmung der Erfolgswahrscheinlichkeit beim Verbindungsaufbau unter Verwendung des in dieser Arbeit vorgestellten Regelwerks.

Um eine verlässliche Evaluation des Verfahrens SYNI sowie des MFB Protokolls durchzuführen, wurde die Evaluation mit der im vorherigen Abschnitt vorgestellten

Testumgebung durchgeführt. Mit Hilfe der Testumgebung ist es möglich eindeutig zu bestimmen, welche Pakete von welchem Router gesendet bzw. weitergeleitet wurden und welche vom NAT Router entsprechend an den internen Rechner weitergeleitet wurden. Führt man die Evaluation in einer realen Umgebung z.B. dem Internet durch, kann nur bestimmt werden, ob eine Verbindung aufgebaut wurde oder nicht. Allerdings ist hierbei nicht ersichtlich, ob beispielsweise der NAT Router von A das Paket nicht an den NAT Router von B weitergeleitet hat oder ob dieser das Paket gefiltert hat.

Nummer	Hersteller	Modell	Firmware Version
1	Netgear	FM114P	Version 1.4 Release 26
2	pfSense	FreeBSD	1.2.3
3	AVM	FRITZ!Box 7050	14.04.33
4	SMC	7004ABR	Runtime Code: 1.00
			Boot Code: V0.15T
5	IPCop	Linux	1.4.20
6	Level One	FBR-1415TX	R1.94p0v
7	D-Link	DI-604 D1	V3.02
8	SMC	7004VBR	Runtime Code: R1.00
			Boot Code: R1.0606.0707
9	Netgear	RP614 v3	V6.0GR
10	Netgear	FVS318	v3.0_26RC1
11	Digitus	DN-11004-O	02.00.00.03
12	Asus	RX3041	Runtime Code: V2.1.2.114
			Boot Code: V0.1.5.24
13	SOHO	IS104A B4	V2.00.0068
14	D-Link	DI-604 B2	1.80
15	D-Link	DIR-100 A1	v1.00DE

Tabelle 3.5: Verwendete NAT Router

Im Rahmen der Evaluation wurde die Testumgebung mit insgesamt 15 verschiedenen NAT Routern verwendet. Die verwendeten Modelle sind in Tabelle 3.5 aufgelistet. Alle NAT Router wurden vor der Evaluation auf die entsprechenden Werkseinstellungen zurückgesetzt und außer der Konfiguration der PPPoE Daten und der Konfiguration der lokalen Netzwerke wurden keine Einstellungen geändert.

Während der Testläufe wurde festgestellt, dass einige der verwendeten NAT Router den TTL Wert nicht ordnungsgemäß dekrementieren. Um sicherzustellen, dass alle

Pakete, die mit einem TTL Wert von 2 gesendet werden, nicht beim Ziel NAT Router ankommen, wurde beim PPPoE Server ein Linux Netzfilter verwendet. Dieser Filter dekrementiert den TTL Wert von jedem Paket um den Wert 3. Hierdurch wird sichergestellt, dass alle Pakete, die mit einem TTL Wert von 2 gesendet werden, den Ziel NAT Router nicht erreichen und eine ICMP Nachricht vom Typ *Zeitlimit überschritten* erzeugt wird. In der Praxis liegen zwischen den NAT Routern R_A und R_B in der Regel mehrere Stationen, so dass Pakete mit einem TTL Wert von 2 ihr Ziel nicht erreichen.

3.5.1 SYNI

Um das TCP Hole Punching Verfahren SYNI zu evaluieren, wurde das Verfahren entsprechend der in Abschnitt 3.4.2 beschriebenen Schnittstellen implementiert. Die Testsoftware für die beiden Kommunikationspartner A und B wurde auf 2 Rechnern mit Windows 7 SP1 ausgeführt. Insgesamt wurden 15 NAT Router getestet, so dass insgesamt $15 \cdot 14 = 210$ Testfälle evaluiert wurden. Die Testergebnisse sind in Tabelle 3.6 zusammengefasst. Durch die mit dem Testsystem erstellten Netzwerkmitschnitte konnte verifiziert werden, dass in allen Testfällen das SYNI Verfahren wie erwartet durchgeführt wurde. Im Folgenden werden die Ursachen für die nicht erfolgreichen Testfälle kurz betrachtet.

In 132 von 210 ausgeführten Testfällen konnte mit dem Verfahren SYNI erfolgreich eine TCP Verbindung hergestellt werden. Alle Testfälle, in denen NAT Router 2 (*pfSense*) oder NAT Router 12 (*Asus RX3041*) involviert waren sind fehlgeschlagen, da bei beiden NAT Routern die verwendete externe Portnummer nicht bestimmt werden konnte. Der Grund hierfür ist, dass beide NAT Router ein Adressen- und Port-abhängiges Portzuordnungsverhalten sowie ein nicht vorhersagbares Portauswahlverhalten haben (siehe Abschnitt 3.5.3). Die Bestimmung der verwendeten externen Portnummer ist für alle Hole Punching Verfahren eine Voraussetzung, die von beiden Routern nicht erfüllt wird.

Betrachtet man diese beiden NAT Router nicht, so ergeben sich $13 \cdot 12 = 156$ Testfälle von denen 132 erfolgreich durchgeführt worden sind $(84,6\%)$. In allen anderen fehlgeschlagenen Testfällen wurden die NAT Router 5 (*IPCop*) oder NAT Router 6 (*Level One FBR-1415TX*) als R_B verwendet. Die Ursache dafür, dass das Verfahren SYNI in Verbindung mit IPCop als R_B nicht erfolgreich war, lag an der Tatsache, dass IPCop ein ausgehendes SYN-ACK Paket nach einem ausgehenden SYN Paket nicht weiterleitet, so dass die Bedingung **B4** nicht erfüllt wird. Ähnlich verhält

A/B	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	-	×	✓	✓	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓
2	×	-	×	×	×	×	×	×	×	×	×	×	×	×	×
3	✓	×	-	✓	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓
4	✓	×	✓	-	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓
5	✓	×	✓	✓	-	×	✓	✓	✓	✓	✓	×	✓	✓	✓
6	✓	×	✓	✓	×	-	✓	✓	✓	✓	✓	×	✓	✓	✓
7	✓	×	✓	✓	×	×	-	✓	✓	✓	✓	×	✓	✓	✓
8	✓	×	✓	✓	×	×	✓	-	✓	✓	✓	×	✓	✓	✓
9	✓	×	✓	✓	×	×	✓	✓	-	✓	✓	×	✓	✓	✓
10	✓	×	✓	✓	×	×	✓	✓	✓	-	✓	×	✓	✓	✓
11	✓	×	✓	✓	×	×	✓	✓	✓	✓	-	×	✓	✓	✓
12	×	×	×	×	×	×	×	×	×	×	×	-	×	×	×
13	✓	×	✓	✓	×	×	✓	✓	✓	✓	✓	×	-	✓	✓
14	✓	×	✓	✓	×	×	✓	✓	✓	✓	✓	×	✓	-	✓
15	✓	×	✓	✓	×	×	✓	✓	✓	✓	✓	×	✓	✓	-

Tabelle 3.6: Testergebnisse von SYNI mit einem TTL Wert von 2

es sich bei NAT Router 6. Dieser sendet zwar die ausgehende Sequenz SYN und SYN-ACK, allerdings wird das anschließend eingehende ACK gefiltert, so dass die Bedingung **B4** nicht erfüllt ist. Solange nicht beide Kommunikationspartner einen NAT Router verwenden, der dieses Verhalten zeigt, kann durch ein Rollentausch von A und B eine Verbindung hergestellt werden. Betrachtet man die Erfolgswahrscheinlichkeit mit Rollentausch und ohne die NAT Router 2 und 12, ergibt sich eine Erfolgswahrscheinlichkeit von $98,7\%$.

In weiteren Versuchen wurde festgestellt, dass der NAT Router 6 ein ausgehendes SYN-ACK erlaubt, wenn zuvor kein ausgehendes SYN gesendet wurde. Anschließend wird auch das eingehende ACK weitergeleitet (SYNI ohne Schritt 2). Allerdings würde das Weglassen des 2. Schritts bei den NAT Routern 1 und 10 dazu führen, dass das Verfahren fehlschlägt, was zu einer insgesamt schlechteren Gesamterfolgswahrscheinlichkeit führt.

Zusätzlich zu den Labortests wurde das Verfahren SYNI in der Praxis getestet. Hierfür wurde das Verfahren zwischen zwei Rechnern, die über zwei verschiedene NAT Router und Internet Service Provider mit dem Internet verbunden waren, durchgeführt. Ein Rechner verwendete einen *T-Com SpeedPort W701V* NAT Router und der zweite Rechner einen *AVM FRITZ!Box 7140* NAT Router. In diesem Szenario

wurde mit dem Netzwerk Diagnose Programm *Traceroute* der entsprechende geringe TTL Wert ermittelt, der vom Rechner A benötigt wird. Beide Rechner haben ihren externen IP Endpunkt mit dem MFB Protokoll bestimmt und waren über einen Rendezvous Server miteinander verbunden. Das Verfahren SYNI wurde in beiden Richtungen erfolgreich durchgeführt.

Auch in diesem Szenario wurde überprüft, ob der 2. Schritt notwendig ist. Wurde der FRITZ!Box 7140 NAT Router auf der Seite von Rechner B eingesetzt, war der 2. Schritt nicht notwendig. Wenn allerdings der NAT Router T-Com SpeedPort W701V auf der Seite von Rechner B einsetzt wurde, war dieser Schritt notwendig, da dieser Router ein einzelnes ausgehendes SYN-ACK nicht erlaubt.

Standard TTL Das SYNI Verfahren kann, wie in Abschnitt 3.2.1 beschrieben, mit einem Standard TTL Wert auf der Seite von A durchgeführt werden. Die Ergebnisse dieser Variante sind in Tabelle 3.7 zusammengefasst.

A/B	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	-	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	×
2	×	-	×	×	×	✓	✓	✓	×	×	×	×	×	✓	×
3	✓	×	-	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	×
4	✓	×	✓	-	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	×
5	✓	×	✓	✓	-	✓	✓	✓	✓	✓	✓	×	✓	✓	×
6	✓	×	✓	✓	✓	-	✓	✓	✓	✓	✓	×	✓	✓	×
7	✓	×	✓	×	✓	✓	-	✓	✓	✓	✓	×	✓	✓	×
8	✓	×	✓	×	✓	✓	✓	-	✓	✓	✓	×	✓	✓	×
9	✓	×	✓	×	✓	✓	✓	✓	-	✓	✓	×	✓	✓	×
10	✓	×	✓	×	✓	✓	✓	✓	✓	-	✓	×	✓	✓	×
11	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	-	×	✓	✓	×
12	×	×	×	×	×	✓	✓	✓	×	×	✓	-	×	✓	×
13	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	-	✓	×
14	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	-	×
15	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	-

Tabelle 3.7: Testergebnisse von SYNI mit einem Standard TTL Wert

Die Ergebnisse dieser Variante unterscheiden sich bei machen Routern von denen aus der ersten Varianten mit geringem TTL Wert. In einigen Testfällen waren Verbindungsversuche erfolgreich, in denen vorher keine Verbindung aufgebaut werden konnte und umgekehrt.

In allen Testfällen, bei denen der NAT Router 15 auf der Seite von B verwendet wurde, konnten in dieser Variante keine Verbindungen hergestellt werden. Der Grund hierfür ist, dass der NAT Router 15 auf das ankommende SYN Paket mit einem TCP RST Paket reagiert. In diesem Fall wird beim NAT Router 15 die Portzuordnung, die zuvor von B ermittelt wurde, ungültig, so dass für das von B ausgehende SYN und anschließende SYN-ACK Paket eine neue Portzuordung erstellt wird. Die Portnummer, die im SYN-ACK Paket, das bei A ankommt, verwendet wird, entspricht somit nicht der erwarteten Portnummer und wird dementsprechend gefiltert.

Zusätzlich zu den Testfällen, in denen der NAT Router 15 als R_B verwendet wurde, sind die Verbindungsversuche bei denen die Router 7, 8, 9 und 10 als R_A und der NAT Router 4 als R_B verwendet wurden fehlgeschlagen. Der NAT Router 4 antwortet wie auch der NAT Router 15 mit einem TCP RST auf das ankommende SYN Paket. In diesen Testfällen sorgt das TCP RST allerdings nicht dafür, dass eine neue Portnummer auf der Seite von R_B verwendet wird, sondern dass die NAT Router 7, 8, 9 und 10 aufgrund des ankommenden RST Pakets alle nachfolgenden Pakete filtern.

Insgesamt hat die Variante mit einem Standard TTL Wert in 16 Fällen, in denen vorher eine Verbindung hergestellt werden konnte, zu einem nicht erfolgreichen Verbindungsaufbau geführt.

Im Gegensatz zu diesen Testfällen, waren einige Testfälle erfolgreich, in denen vorher keine Verbindung hergestellt werden konnte. Alle Testfälle, bei denen der NAT Router 5 auf der Seite von B verwendet wurde, führten zu einem erfolgreichen Verbindungsaufbau mit Ausnahme der Testfälle in denen die NAT Router 2 oder 12 auf der Seite von A verwendet wurden. In allen diesen Testfällen wurde die Verbindung aufgebaut, da das SYN Paket, welches von A gesendet wird, von R_B an B weitergeleitet wird. Der NAT Router 5 akzeptiert nach einen ausgehenden SYN Paket zwar kein ausgehendes SYN-ACK Paket, dafür aber ein eingehendes SYN Paket. Der TCP Socket auf der Seite von A schickt insgesamt 3 SYN Pakete, so dass eines dieser wiederholten SYN Pakete B erreicht, nachdem B ein ausgehendes SYN Paket gesendet hat. Anschließend wird auch das ausgehende SYN-ACK von diesem Router weitergeleitet, da dieses Paket erwartungsgemäß auf ein eingehendes SYN Paket folgt. Die Testfälle mit den NAT Routern 2 und 12 waren trotz dieses Verhaltens nicht erfolgreich, weil beide NAT Router ein Endpunkt-abhängiges Portzuordnungsverhalten haben. Hierdurch werden die Pakete nicht von der von R_A erwarteten Portnummer aus gesendet, so dass die Pakete gefiltert werden.

Die Verbindungsversuche, bei denen der NAT Router 6 auf der Seite von B verwendet wurde, waren in dieser Variante im Gegensatz zu der mit einem geringen TTL Wert, erfolgreich. Der Grund hierfür ist, dass der NAT Router 6 ein Endpunktunabhängiges Filterverhalten zeigt. Durch die bei der Bestimmung des externen IP Endpunkts erstellten Portzuordnung kann somit jeder beliebige Rechner Pakete an den internen Rechner senden. Hierdurch wird das erste SYN Paket, welches A an B sendet, von R_B weitergeleitet und die Verbindung aufgebaut, ohne dass das Verfahren SYNI wirklich ausgeführt werden muss.

Dieses Endpunkt-unabhängige Filterverhalten wurde, wie in Abschnitt 3.5.3 gezeigt wird, ebenfalls bei den NAT Routern 6, 7, 8 und 14 festgestellt. Aufgrund dessen konnte auch in den Fällen, in denen die NAT Router 2 und 12 als R_A und die NAT Routern 6, 7, 8 und 14 als R_B verwendet wurden eine Verbindung hergestellt werden.

Eine weitere Kombination, die in dieser Variante zu einer erfolgreichen Verbindung führte, ist die in der NAT Router 12 als R_A und NAT Router 11 als R_B verwendet wurden. Der NAT Router 11 antwortet auf das erste gesendete SYN Paket mit einem RST Paket. In diesem Fall sorgt das RST Paket bei NAT Router 12 dafür, dass nach diesem Paket alle weiteren Pakete an den internen Rechner weitergeleitet werden. Im Gegensatz zu den vorherigen Fällen, in denen ein RST Paket den Verbindungsaufbau negativ beeinflusst hat, hat das RST Paket in diesem Fall dafür gesorgt, dass eine Verbindung aufgebaut werden konnte.

Insgesamt hat die Variante mit einem Standard TTL Wert in 33 Fällen, in denen vorher keine Verbindung hergestellt werden konnte, zu einem erfolgreichen Verbindungsaufbau geführt. Subtrahiert man nun die 16 Testfälle, bei denen in dieser Variante keine Verbindung aufgebaut werden konnte, so ist diese Variante insgesamt um 17 Testfälle erfolgreicher. Allerdings ist in den zusätzlich erfolgreichen Testfällen das Verfahren nicht wie spezifiziert durchgeführt worden, sondern führte aus anderen Gründen (z.B. Endpunkt-unabhängige Filterverhalten) zum Erfolg.

3.5.2 Verwandte Arbeiten

Zusätzlich zur Evaluation des in dieser Arbeit vorgestellten TCP Hole Punching Verfahrens SYNI, wurde das Verfahren NUTSS (b) evaluiert. NUTSS (b) wurde als einziges Verfahren der verwandten Arbeiten evaluiert, da dieses Verfahren Bestandteil des in Abschnitt 3.3.4 vorgestellten Regelwerks ist und somit auch für die Evaluation des Regelwerks benötigt wird. Das Verfahren wurde, wie in Abschnitt 3.2.2

A/B	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	-	×	×	×	×	×	×	×	×	×	×	×	×	×	×
2	×	-	×	×	×	×	×	×	×	×	×	×	×	×	×
3	✓	×	-	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓
4	✓	×	✓	-	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓
5	✓	×	✓	✓	-	✓	✓	✓	✓	✓	✓	×	✓	✓	✓
6	×	×	×	×	×	-	×	×	×	×	×	×	×	×	×
7	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	✓	✓	✓	✓
8	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	✓	✓	✓
9	✓	×	✓	✓	✓	✓	✓	✓	-	✓	✓	×	✓	✓	✓
10	✓	×	✓	✓	✓	✓	✓	✓	✓	-	✓	×	✓	✓	✓
11	×	×	×	×	×	×	×	×	×	×	-	×	×	×	×
12	×	×	×	×	×	×	×	×	×	×	×	-	×	×	×
13	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	-	✓	✓
14	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓
15	×	×	×	×	×	×	×	×	×	×	×	×	×	×	-

Tabelle 3.8: Testergebnisse von NUTSS(b)

beschrieben, implementiert und mit Hilfe der Testumgebung evaluiert. NUTSS (b) wurde in 114 von 210 Testfällen erfolgreich durchgeführt (54, 3%). Die Ergebnisse der Evaluation sind in Tabelle 3.8 zusammengefasst. Im Folgenden werden die einzelnen Testfälle näher betrachtet.

Die Testfälle, in denen die NAT Router mit den Nummern 1, 2, 6, 7, 12 und 15 auf der Seite von A verwendet wurden, sind fehlgeschlagen, da die Router die Sequenz *ausgehendes SYN eingehendes SYN* nicht akzeptieren. Hierdurch kann die Verbindung, die von B initiiert wird, nicht erfolgreich aufgebaut werden. Waren die NAT Router 3, 4, 5, 9, 10 und 13 auf der Seite von A, waren nur die Verbindungen nicht erfolgreich, bei denen die NAT Routern 2 und 12 auf der Seite von B eingesetzt wurden. Der Grund hierfür ist der, dass die NAT Router 2 und 12 ein Adressen- und Port-abhängiges Portzuordnungsverhalten sowie ein nicht vorhersagbares Portauswahlverhalten haben (siehe Abschnitt 3.5.3). Hierdurch wird für den Verbindungsaufbau, der von B initiiert wird, eine neue Portnummer verwendet, die nicht mit der übereinstimmt, die *A* verwendet hat um die Portzuordnung zu erstellen. Die NAT Router 7, 8 und 14 haben unabhängig von der verwendeten Portnummer alle TCP Pakete weitergeleitet, so dass alle Testfälle in denen diese NAT Router auf der Seite von A verwendet wurden, erfolgreich durchgeführt wurden.

NUTSS (b) wurde zusätzlich in einer weiteren Variante evaluiert. Hierbei wurde das SYN Paket, das von A gesendet wird, mit einem Standard TTL Wert gesendet. In 9 Testfällen, in denen mit einem geringen TTL Wert eine Verbindung aufgebaut werden konnte, waren in dieser Variante nicht erfolgreich. Der Grund hierfür ist, dass aufgrund des SYN Pakets ein TCP RST von der Gegenstelle gesendet wurde. In 3 Testfällen hat dieses RST Paket allerdings dafür gesorgt, dass eine Verbindung aufgebaut werden konnte, die in der Variante mit einem geringen TTL Wert nicht hergestellt werden konnte. Dieses Verhalten wurde auch bei der Evaluation des Verfahrens SYNI beobachtet. Insgesamt wurden in dieser Variante 6 Verbindungen weniger aufgebaut.

3.5.3 MFB Protokoll

Die im Folgenden vorgestellte Evaluation des MFB Protokolls gliedert sich in zwei Teile. Im ersten Teil werden verschiedene NAT Router mit dem MFB Protokoll auf ihre Eigenschaften getestet. Basierend auf diesen Eigenschaften wird im zweiten Teil das in Abschnitt 3.3.4 vorgestellte Regelwerk bezüglich der Erfolgswahrscheinlichkeit beim Verbindungsaufbau evaluiert.

NAT Charakteristiken

In diesem Abschnitt werden die Ergebnisse, die mit dem MFB Protokoll ermittelt wurden, dargestellt. Hierfür wurden die Client und Server Komponenten des MFB Protokolls, wie in Abschnitt 3.3 beschrieben, unter Verwendung der Programmiersprache .NET C# implementiert. Zusätzlich wurde das Framework WinPcap [17] eingesetzt. WinPcap stellt einen Raw Socket zur Verfügung, der es erlaubt eigene TCP Pakete zu senden und alle eingehenden Pakete mitzulesen.

Die Evaluation wurde mit dem in diesem Kapitel vorgestellten Testsystem sowie zusätzlich als öffentlicher Test durchgeführt. Um den öffentlichen Test durchzuführen wurde ein MFB Server eingerichtet, der über zwei öffentliche IP Adressen verfügt und auf jeweils zwei Portnummern erreichbar ist. Die daraus resultierenden IP Endpunkte wurden in die Client Software, die an die teilnehmenden Tester verteilt wurde, fest eingetragen. Die Ergebnisse der Tests wurden von den MFB Clients anschließend an einen Datenbank Server gesendet.

Insgesamt wurden 40 eindeutige Tests von den teilnehmenden Nutzern durchgeführt. Im Folgenden werden die Testergebnisse des öffentlichen Tests sowie die Ergebnisse, die mit dem Testsystem ermittelt wurden, zusammengefasst dargestellt.

Portzuordnungsverhalten	Labor		Internet		Gesamt	
	UDP	TCP	UDP	TCP	UDP	TCP
Endpunkt-unabhängig	80%	66,7%	72,5%	80%	74,5%	76,4%
Adressen-abhängig	0%	0%	0%	0%	0%	0%
Port-abhängig	0%	0%	0%	0%	0%	0%
Adressen- und Port-abhängig	20%	33,3%	27,5%	20%	25,5%	23,6%

Tabelle 3.9: Portzuordnungsverhalten für TCP und UDP

In Tabelle 3.9 sind die Ergebnisse des Portzuordnungsverhaltens aller getesteten NAT Router zusammengefasst. Hierbei wird deutlich, dass die Mehrheit der NAT Router ein Endpunkt-unabhängiges Portzuordnungsverhalten zeigt. Die ermittelten Ergebnisse decken sich mit den Ergebnissen, die von Guha et al. in [32] veröffentlicht wurden. Die Tatsache, dass die meisten NAT Router ein Endpunkt-unabhängiges Portzuordnungsverhalten zeigen, ist von Vorteil für die in dieser Arbeit präsentierten Hole Punching Verfahren.

Ein weiterer interessanter Punkt dieser Ergebnisse ist, dass einige NAT Router ein unterschiedliches Verhalten für TCP und für UDP zeigen. Im Allgemeinen ist das Portzuordnungsverhalten für TCP restriktiver als das für UDP.

Filterverhalten	Labor		Internet		Gesamt	
	UDP	TCP	UDP	TCP	UDP	TCP
Endpunkt-unabhängig	46,7%	26,7%	10%	0%	20%	7,3%
Adressen-abhängig	13,3%	0%	0%	0%	3,6%	0%
Port-abhängig	0%	0%	0%	0%	0%	0%
Adressen- und Port-abhängig	40%	73,3%	90%	100%	76,4%	92,7%

Tabelle 3.10: Filterverhalten für TCP und UDP

Das Filterverhalten der getesteten Router ist in Tabelle 3.10 zusammengefasst. Im Allgemeinen ist das Filterverhalten restriktiver als das Portzuordnungsverhalten. Die meisten Router haben ein Adressen- und Port-abhängiges Filterverhalten für UDP (76,4%) und TCP (92,7%).

Die ermittelten Ergebnisse des Filterantwortverhaltens sind in Tabelle 3.11 zusammengefasst. Die Mehrheit der getesteten Router sendet keine Antwort, wenn ein Paket empfangen wird für das keine Portzuordnung existiert oder das aufgrund

Filterantwortverhalten	Labor		Internet		Gesamt	
	UDP	TCP	UDP	TCP	UDP	TCP
Keine Antwort	80%	73,3%	72,5%	80%	76,4%	80%
TCP RST	-	20%	-	5%	-	9,1%
ICMP	20%	6,7%	27,5%	15%	23,6%	10,9%

Tabelle 3.11: Filterantwortverhalten für TCP und UDP

des Filterverhaltens nicht weitergeleitet wurde. Dieses Verhalten ist von Vorteil für die Verfahren SYNI und NUTSS(b), da hierdurch das erste SYN Paket mit einem Standard TTL Wert gesendet werden kann, ohne dass der Ziel NAT Router eine Fehlernachricht sendet.

Weitere Eigenschaften	Labor		Internet		Gesamt	
	UDP	TCP	UDP	TCP	UDP	TCP
Vorhersagbare Portauswahl	93,3%	86,7%	97,5%	97,5%	96,4%	94,5%
Porterhaltung	40%	46,7%	80%	80%	69,1%	70,9%
Hairpinning	13,3%	26,7%	10%	2,5%	10,9%	9,1%
Modifikation der TCP Seq.	-	0%	-	0%	-	0%
Unterstützung von UPnP	13,3%		10%		10,9%	

Tabelle 3.12: Weitere Eigenschaften der NAT Router für TCP und UDP

In Tabelle 3.12 sind weitere Eigenschaften der getesteten NAT Router zusammengefasst. Die Ergebnisse zeigen, dass die Mehrheit der getesteten NAT Router ein vorhersagbares Portauswahlverhalten zeigt. Dieses Verhalten resultiert weitestgehend aus der Tatsache, dass die meisten NAT Router ein Endpunkt-unabhängiges Portzuordnungsverhalten zeigen. Zusätzlich unterstützen die meisten NAT Router ebenfalls die Eigenschaft der Porterhaltung. Lediglich zwei Router zeigten ein völlig zufälliges Portauswahlverhalten für UDP und drei NAT Router für TCP, so dass die verwendete Portnummer praktisch nicht vorhergesagt werden kann.

Ford hat 2005 in [27] die Notwendigkeit von Hairpinning in Verbindung mit NAT Traversal Verfahren und mehreren hintereinander liegenden NAT Routern herausgestellt. Allerdings zeigen die Ergebnisse, dass kaum einer der getesteten NAT Router Hairpinning unterstützt. Von den getesteten NAT Routern unterstützte nur ein NAT Router Hairpinning für TCP und UDP. Vier weitere Router zeigten Hairpinning Unterstützung für UDP oder für TCP.

Eine Bedingung des Verfahrens SYNI ist, dass der verwendete NAT Router die TCP Sequenznummern nicht modifiziert. Die Ergebnisse der Evaluation zeigen, dass keiner der getesteten NAT Router dieses Verhalten zeigt.

Die letzte Eigenschaft, die ermittelt wurde, ist die Unterstützung für UPnP. Die Ergebnisse des UPnP Tests aus dem öffentlichen Test zeigen, dass mit 10% der NAT Router das Erstellen einer Portweiterleitung per UPnP möglich war. Hierbei ist allerdings nicht ersichtlich, ob die anderen Router UPnP nicht unterstützen oder ob UPnP deaktiviert war. Bei den Routern, die in dem Testsystem getestet wurden, konnte hingegen beides ermittelt werden. Insgesamt unterstützen 12 von 15 Routern UPnP. Allerdings konnte nur mit den NAT Routern 1 und 13 erfolgreich eine Portweiterleitung erstellt werden. Dieses Ergebnis zeigt die Notwendigkeit von Hole Punching Verfahren, da UPnP nicht immer verwendet werden kann.

Nachdem die durch das MFB Protokoll bestimmten Eigenschaften der NAT Router dargestellt wurden, wird im Folgenden kurz das Verhalten der Sockets beschrieben. Die Eigenschaften, die bestimmt wurden, resultieren aus den Anforderungen der NAT Traversal Verfahren SYNI und NUTSS(b).

Das Verfahren NUTSS(b) setzt voraus, das der verwendete Socket kein RST Paket sendet, wenn dieser geschlossen wird. Daher wurde das Socketverhalten unter Verwendung der Betriebssysteme Windows XP SP3 und Windows 7 bestimmt. In beiden Fällen wurde kein RST gesendet, wenn der Socket geschlossen wurde (siehe Tabelle 3.13).

Socketverhalten	Windows XP SP3	Windows 7
RST nach Schließen	nein	nein
TTL Modifikation	nicht möglich	nach Handshake

Tabelle 3.13: Socketverhalten unter verschiedenen Betriebssystemen

Die zweite Socket Eigenschaft, die bestimmt wurde, resultiert aus der Anforderung des Verfahrens SYNI. Das Verfahren SYNI setzt voraus, dass der TTL Wert nach dem Verbindungsaufbau (Handshake) modifiziert werden kann. Bei dieser Eigenschaft zeigten die Betriebssysteme Windows 7 und Windows XP SP3 ein unterschiedliches Verhalten. Unter Windows 7 konnte der TTL Wert nach dem Verbindungsaufbau erfolgreich modifiziert werden. Windows XP SP3 hingegen erlaubte ebenso die Modifikation, allerdings wurde der TTL Wert des Sockets ignoriert. Hierdurch werden alle Pakete immer mit demselben Standard TTL Wert gesendet, so dass es unter Windows XP nicht möglich ist einen anderen TTL Wert zu verwen-

den. Das bedeutet, dass die NAT Traversal Verfahren unter Windows XP nur in der Variante mit Standard TTL verwendet werden können.

Zusammenfassend zeigen die Ergebnisse, dass NAT Router kein einheitliches Verhalten zeigen. Allerdings können in den meisten Fällen NAT Traversal Verfahren eingesetzt werden, um eine Verbindung zwischen zwei Kommunikationspartnern herzustellen.

Regelwerk

Nachdem im vorherigen Abschnitt die Eigenschaften der getesteten NAT Router betrachtet wurden, wird in diesem Abschnitt das Regelwerk zur Auswahl des zu verwendenden NAT Traversal Verfahrens evaluiert.

A/B	1	2	3	4	5	6	7	8
1	-	U	U	U	U	U	U	U
2	U_U	-	M	DU_U	M	DT	DT	DT
3	U_U	M	-	S	$SSTTL$	DT	DT	DT
4	U_U	DU_U	S	-	$SSTTL$	DT	DT	DT
5	U_U	M	S	S	-	DT	DT	DT
6	U_U	DT_U	DT_U	DT_U	DT_U	-	DT	DT
7	U_U	DT_U	DT_U	DT_U	DT_U	DT	-	DT
8	U_U	DT_U	DT_U	DT_U	DT_U	DT	DT	-
9	U_U	DU_U	S	S	$SSTTL$	DT	DT	DT
10	U_U	M	S	S	$SSTTL$	DT	DT	DT
11	U_U	DU_U	S	S	$SSTTL$	DT	DT	DT
12	U_U	M	M	DU	M	DT	DT	DT
13	U	U	U	U	U	U	U	U
14	U_U	DT_U	DT_U	DT_U	DT_U	DT	DT	DT
15	U_U	DU_U	S	S	$SSTTL$	DT	DT	DT

U	= UPnP	U_U	= UPnP (Umkehr)
DT	= Direkt TCP	DT_U	= Direkt TCP (Umkehr)
DU	= Direkt UDP	DU_U	= Direkt UDP (Umkehr)
S	= SYNI	$SSTTL$	= SYNI mit Standard TTL
M	= Mittelsmann		

Tabelle 3.14: Testergebnisse des Regelwerks - Teil 1

A/B	9	10	11	12	13	14	15
1	U	U	U	U	U	U	U
2	DU	M	DU_U	M	U_U	DT	DU
3	$SSTTL$	$SSTTL$	$SSTTL$	M	U_U	DT	S
4	$SSTTL$	$SSTTL$	$SSTTL$	DU_U	U_U	DT	S
5	$SSTTL$	$SSTTL$	$SSTTL$	M	U_U	DT	S
6	DT_U	DT_U	DT_U	DT_U	U_U	DT	DT_U
7	DT_U	DT_U	DT_U	DT_U	U_U	DT	DT_U
8	DT_U	DT_U	DT_U	DT_U	U_U	DT	DT_U
9	-	$SSTTL$	$SSTTL$	DU_U	U_U	DT	S
10	$SSTTL$	-	$SSTTL$	M	U_U	DT	S
11	$SSTTL$	$SSTTL$	-	DU_U	U_U	DT	S
12	DU	M	DU	-	U_U	DT	DU_U
13	U	U	U	U	-	U	U
14	DT_U	DT_U	DT_U	DT_U	U_U	-	DT_U
15	$SSTTL$	$SSTTL$	$SSTTL$	DU_U	U_U	DT	-

U	= UPnP	U_U	= UPnP (Umkehr)
DT	= Direkt TCP	DT_U	= Direkt TCP (Umkehr)
DU	= Direkt UDP	DU_U	= Direkt UDP (Umkehr)
S	= SYNI	$SSTTL$	= SYNI mit Standard TTL
M	= Mittelsmann		

Tabelle 3.15: Testergebnisse des Regelwerks - Teil 2

Die beiden Kommunikationspartner A und B wurden auf zwei Rechnern mit Windows 7 ausgeführt. A und B konnten jeweils einen durch WinPcap zur Verfügung gestellten Raw Socket verwenden. In den Tabellen 3.14 und 3.15 sind die jeweils vom Regelwerk ausgewählten Verfahren dargestellt.

Insgesamt konnten in 196 von 210 Verbindungsversuchen im ersten Versuch eine direkte Verbindung per TCP oder UDP/RUDP aufgebaut werden. In 14 Fällen konnte keine direkte Verbindung aufbaut werden, so dass in diesen Fällen nur über einen Mittelsmann kommuniziert werden konnte.

3.5.4 Sonderfälle

Während der Evaluation der Hole Punching Verfahren und des MFB Protokolls wurde eine Reihe von speziellen Eigenschaften bei einigen NAT Routern beobachtet.

Die NAT Router 6, 7 und 8 akzeptieren nach einem ausgehenden UDP Paket alle Pakete, die an die durch das UDP Paket erstellte Portzuordnung gesendet werden. Dieses Verhalten ist in Abbildung 3.13 grafisch dargestellt.

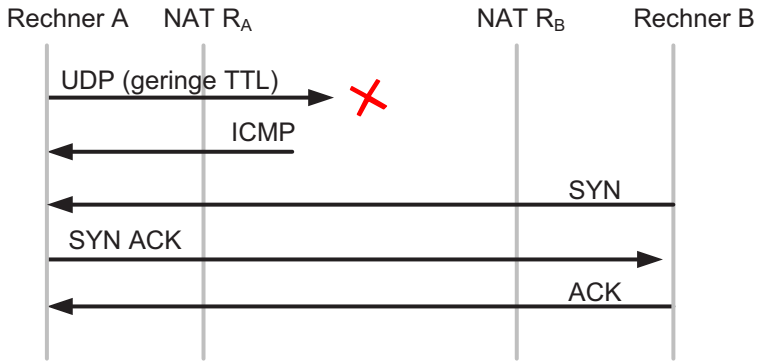


Abbildung 3.13: Ausgehendes UDP Paket und eingehende TCP Verbindung

Der NAT Router 11 zeigte als einziger Router ein spezielles Verhalten bei einem eingehenden TCP RST Paket. Die Sequenz ausgehendes SYN, eingehendes SYN wird nicht akzeptiert, allerdings erlaubt der NAT Router die Sequenz ausgehendes SYN, eingehendes RST, eingehendes SYN. Das TCP RST Paket gibt im Normalfall einen schwerwiegenden Fehler an, allerdings sorgt das RST Paket beim Digitus DN-11004-O dafür, dass der Verbindungszustand zurückgesetzt wird, aber die Portzuordnung bestehen bleibt. Diese Portzuordnung kann im Folgenden verwendet werden, um eine eingehende TCP Verbindung herzustellen (siehe Abbildung 3.14).

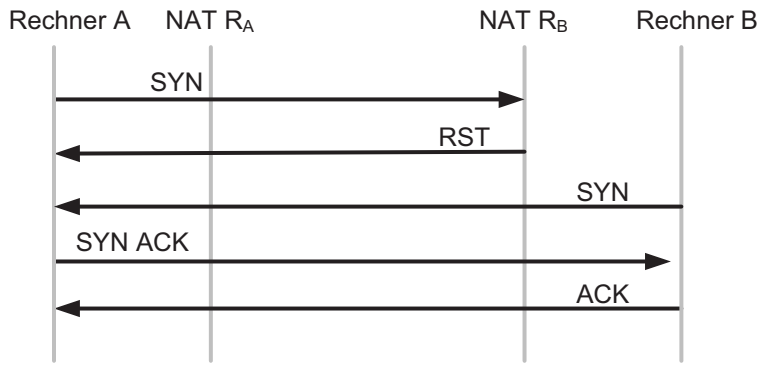


Abbildung 3.14: Eingehendes RST Paket setzt Verbindungszustand zurück

Der NAT Router 15 zeigte ebenso wie NAT Router 11 als einziger Router ein spezielles Verhalten bei einem TCP RST Paket. Empfängt NAT Router 15 ein RST Paket auf einer bestimmten Portnummer, so wird die Portzuordnung für diese Portnummer entfernt. Werden weitere Pakete gesendet, so wird für diese eine neue Portzuordnung mit einer neuen Portnummer erstellt.

3.6 Zusammenfassung

In diesem Kapitel wurden das TCP Hole Punching Verfahren SYNI sowie das MFB Protokoll vorgestellt.

Das vorgestellte TCP Hole Punching Verfahren basiert auf der Injektion eines SYN Pakets zur Initiierung einer neuen TCP Verbindung. In Abschnitt 3.2.2 wurden die Anforderungen des SYNI Verfahrens mit den Anforderungen bereits existierender TCP Hole Punching Verfahren verglichen. Aufgrund der verschiedenen Anforderungen kann mit dem Verfahren SYNI bei bestimmten NAT Routern eine Verbindung hergestellt werden, in denen andere Verfahren keine Verbindung herstellen können.

Im zweiten Teil dieses Kapitels wurde das MFB Protokoll vorgestellt. Das MFB Protokoll ermittelt Informationen über die Netzwerkumgebung eines Rechners wie beispielsweise das Portzuordnungs- und Filterverhalten des NAT Routers. Die durch das MFB Protokoll ermittelten Eigenschaften wurden durch eine Analyse verschiedener NAT Traversal Verfahren bestimmt. Um bereits existierende STUN Server nutzen zu können, wurde das MFB Protokoll so entwickelt, dass eine limitierte Version des MFB Protokolls mit existierenden STUN Servern ausgeführt werden kann. Zusätzlich wurde das MFB Protokoll so entwickelt, dass der MFB Dienst in einem dezentralen Peer-to-Peer System von den Peers angeboten werden kann. Aufbauend auf den durch das MFB Protokoll ermittelten Eigenschaften wurde ein Regelwerk entwickelt, mit dem in einem konkreten Szenario das NAT Traversal Verfahren bestimmt werden kann, dass die höchste Erfolgswahrscheinlichkeit hat.

Um eine verlässliche Evaluation der TCP Hole Punching Verfahren und des MFB Protokolls zu gewährleisten, wurde ein spezielles Testsystem entwickelt. Das Testsystem erstellt für die Kommunikation zwischen den Routern und Rechnern Netzwerkmitschnitte. Hierdurch kann eine eindeutige Aussage darüber getroffen werden, ob ein Verfahren aufgrund seiner Eigenschaften oder durch das zufällige Zusammentreffen von bestimmten Umständen erfolgreich war. In Abschnitt 3.5 wurde gezeigt, dass das Verfahren SYNI eine Erfolgswahrscheinlichkeit von $98,7\%$ erreicht, wenn

man nur die NAT Router betrachtet bei denen man den verwendeten externen IP Endpunkt verlässlich bestimmen kann. Die Evaluation des in diesem Kapitel vorgestellten Regelwerk zeigt, dass in $93,3\%$ der Testfälle im ersten Versuch eine direkte Verbindung hergestellt werden konnte. In den anderen Fällen konnte aufgrund der verwendeten NAT Router keine direkte Verbindung hergestellt werden, so dass in diesen Fällen nur eine Kommunikation per Mittelsmann möglich war.

Kapitel 4

Architektur für komplexe Peer-to-Peer Systeme

In diesem Kapitel wird eine Softwarearchitektur für Forschung und Entwicklung von komplexen Peer-to-Peer Systemen vorgestellt [37]. Eine komplette Peer-to-Peer Applikation muss verschiedene Aufgaben wie NAT Traversal, Bootstrapping, Verbindungsmanagement, Routing, Datenspeicherung und Sicherheit erfüllen. In der vorgestellten Softwarearchitektur wird eine Peer-to-Peer Applikation in verschiedene Schichten und Komponenten unterteilt. Das Grundgerüst einer Peer-to-Peer Applikation kann so durch das Zusammensetzen verschiedener existierender Schichten und Komponenten erstellt werden. Hierdurch wird eine hohe Wiederverwendbarkeit und Austauschbarkeit ermöglicht. Zusätzlich ermöglicht es die vorgestellte Architektur den gleichen Quelltext für die Release Software als auch für eine diskrete Ereignis-basierte Simulation zu verwenden. Dies wird durch das in Abschnitt 2.11 vorgestellte Programmiermodell Gears4Net ermöglicht. Des Weiteren unterstützt die Softwarearchitektur mehrere zeitgleich aktive Peer-to-Peer Netzwerke, so dass verschiedene Netzwerke für unterschiedliche Aufgaben verwendet werden können.

Im Folgenden wird die Architektur, die im weiteren Verlauf als *peers@play Architektur* bezeichnet wird, vorgestellt. Bevor die Architektur im Detail betrachtet wird, wird eine Reihe von Anforderungen beschrieben, die die Softwarearchitektur erfüllen muss. In Abschnitt 4.4 wird gezeigt, dass die peers@play Architektur nur einen geringen Overhead in Bezug auf die Geschwindigkeit und den Speicherverbrauch der Anwendung verursacht. Abschließend werden die für die peers@play Architektur relevanten verwandten Arbeiten vorgestellt sowie die Ergebnisse dieses Kapitels zusammengefasst.

4.1 Anforderungen

Während der Entwicklung des peers@play Projektes zeigte sich, dass die Entwicklung einer komplexen Peer-to-Peer Anwendung mehr erfordert als das Bilden einer Netzwerkstruktur und eines Routing Algorithmus. Auch wenn die Netzwerkstruktur in den meisten Fällen das Hauptaugenmerk des wissenschaftlichen Interesses ist, sind Aufgaben, wie beispielsweise das Bootstrapping, NAT Traversal und die Verbindungsverwaltung und deren Zusammenwirken nicht trivial.

Eine vollständige Peer-to-Peer Applikation, die auf verschiedenen Rechnern hinter NAT Routern mit Breitband Internetzugang und variierenden Bandbreiten und Latenzen ausgeführt wird zu entwickeln, zu verteilen und zu testen ist deutlich aufwändiger als eine neue Netzwerkstruktur zu simulieren. Die hierbei auftretenden Einflüsse können den Unterschied zwischen einer Anwendung, die benutzbar ist und einer Anwendung, die zu langsam oder zu instabil ist, ausmachen. Wenn beispielsweise aufgrund einer hohen Interaktionsgeschwindigkeit innerhalb der Anwendung eine Antwort in 100-150ms erwartet wird, die Anwendung aber einige 100ms benötigt, um dafür eine Verbindung mittels Hole Punching herzustellen, ist dies ein entscheidender Faktor bezüglich der Nutzbarkeit. Daher sind Tests in der realen Umgebung notwendig, um die Umsetzbarkeit einer Applikation zu demonstrieren. Das gilt insbesondere, wenn diese so empfindlich bezüglich der Latenz, Verfügbarkeit und Sicherheit ist, wie eine Peer-to-Peer-basierte Online Welt.

Allerdings existiert zusätzlich zu den Tests in der realen Umgebung die Notwendigkeit Tests und Messungen mit reproduzierbaren Ergebnissen durchzuführen. Der allgemeine Ansatz hierfür ist eine eigene Simulation der zu testenden Protokolle durchzuführen. Um diese Simulation mit Hilfe von Netzwerksimulatoren durchzuführen, muss in den meisten Fällen die zu testende Anwendung neu implementiert werden, um die Anforderungen des jeweiligen Netzwerksimulators zu erfüllen. Infolgedessen werden zwei verschiedene Implementationen desselben Konzepts benötigt: eine für die Simulation und eine für die Release Software. Um dies zu umgehen ist die erste Anforderung, die an die peers@play Architektur gestellt wird, dass die gleiche Implementation für Messungen in einer Simulation als auch für die Release Software verwendet werden kann. Hierbei ist es wichtig, dass die Umsetzung dieser Anforderung nur geringen Overhead in Bezug auf die Verarbeitungszeit und den Speicherverbrauch der Anwendung verursacht, da es sonst nicht möglich ist eine große Anzahl an Peers zu simulieren.

Des Weiteren existieren häufig verschiedene Lösungen bzw. Ansätze für eine Aufgabe (z.B. Bootstrapping Verfahren [46] oder Routing Protokolle [47, 71, 74]). Um verschiedene Ansätze im Gesamtsystem testen zu können, ist es notwendig, dass diese Ansätze einfach austauschbar sind und mit bereits implementierten Komponenten zusammen kombiniert werden können. Das ermöglicht es beispielsweise eine neue Netzwerkstruktur mit bereits bestehenden Komponenten für die Verbindungsverwaltung und das Bootstrapping zu kombinieren und zu testen. Die zweite Anforderung an die peers@play Architektur ist daher die Austauschbarkeit und Wiederverwendbarkeit von existierenden Lösungen.

Im peers@play Projekt werden verschiedene Netzwerkstrukturen (Peer-to-Peer Netzwerke) für verschiedene Aufgaben verwendet. Ein Peer-to-Peer Netzwerk wird beispielsweise als Grundlage für einen Datenspeicher verwendet, der Informationen über die Objekte der virtuellen Welt im 2- oder 3-dimensionalen Raum zur Verfügung stellen muss. Dieses Peer-to-Peer Netzwerk sollte nur beim Eintreten und Verlassen von Peers die Struktur anpassen, da bei jeder Strukturanpassung Teile der gespeicherten Daten zwischen Peers transferiert werden müssen. Das erhöht zum einen die Last innerhalb des Netzwerks und zum anderen erhöht sich die Wahrscheinlichkeit, dass Daten verloren gehen. Im Gegensatz dazu benötigt ein Peer-to-Peer Netzwerk, das die dynamischen Positionen der Nutzer innerhalb der virtuellen Welt abbildet, ein hoch dynamisches Netzwerk. In diesem Fall ändert sich gegebenenfalls mit jeder Positionsänderung eines Nutzers die Netzwerkstruktur. Die peers@play Architektur muss daher mehrere zeitgleich aktive Peer-to-Peer Netzwerke unterstützen, die dieselben grundlegenden Komponenten verwenden. Hierbei ist es wichtig, dass kein Peer-to-Peer Netzwerk in der Lage sein darf so viele Daten zu senden, dass andere Netzwerke dadurch keine Daten mehr senden können. Wäre ein Peer-to-Peer Netzwerk dazu in der Lage, würde dies die Leistung des Gesamtsystems verringern. Die dritte und letzte Anforderung an die Architektur ist daher die Unterstützung mehrerer aktiver Peer-to-Peer Netzwerke.

Zusammenfassend muss die peers@play Architektur die folgenden Anforderungen erfüllen:

R1: Gleicher Quelltext für Simulationen und Release Software mit geringem Overhead in Bezug auf die Verarbeitungszeit und den Speicherverbrauch

R2: Wiederverwendbarkeit und Austauschbarkeit

R3: Unterstützung mehrerer aktiver Peer-to-Peer Netzwerke

4.2 Architektur

Nachdem im vorherigen Abschnitt die an die peers@play Architektur gestellten Anforderungen beschrieben wurden, wird im Folgenden die Architektur näher betrachtet. Basierend auf dem Ansatz von Aberer [5] (siehe Abbildung 4.1) und etablierten Ansätzen im Design von verteilten Systemen, unterteilt die peers@play Architektur eine Applikation in verschiedene Schichten und Komponenten (siehe Abbildung 4.2).

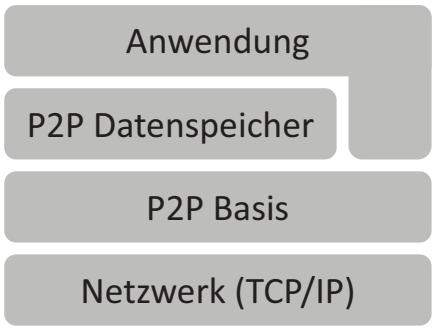


Abbildung 4.1: Schichtenarchitektur von Aberer [5]

Vergleichbar mit anderen Peer-to-Peer Architekturen, wie beispielsweise der von Aberer, beinhaltet die peers@play Architektur eine Anwendungsschicht, eine Datenspeicherungsschicht und eine Schicht für das Peer-to-Peer Netzwerk.

Die Anwendungsschicht in der peers@play Architektur beinhaltet nicht nur die eigentliche Applikationslogik und die Integration zur Visualisierung, sondern kapselt auch Funktionalitäten für typische wiederkehrende Probleme, wie die Gruppenkommunikation und Auswahlprotokolle.

Die Datenspeicherungsschicht ist dafür zuständig Daten auf den teilnehmenden Peers zu speichern. Zusätzlich kann die Datenspeicherungsschicht dafür sorgen, dass die Daten auf mehreren Peers repliziert werden, um so eine höhere Verfügbarkeit zu gewährleisten. Um Daten zu speichern stellt die Datenspeicherungsschicht im einfachsten Fall eine rudimentäre Schnittstelle [21] zur Verfügung mit der Daten unter einem bestimmten Schlüssel gespeichert und angefragt werden können. Hier ist es auch möglich speziellere Schnittstellen zu verwenden, die beispielsweise Versions- oder Zugriffskontrolle bieten. Allerdings reduziert dies die Austauschbarkeit der verschiedenen Ansätze innerhalb dieser Schicht.

Die Schicht des Peer-to-Peer Netzwerks ist verantwortlich für die Netzwerkstruktur der verbundenen Peers sowie für das Routing von Nachrichten innerhalb dieser

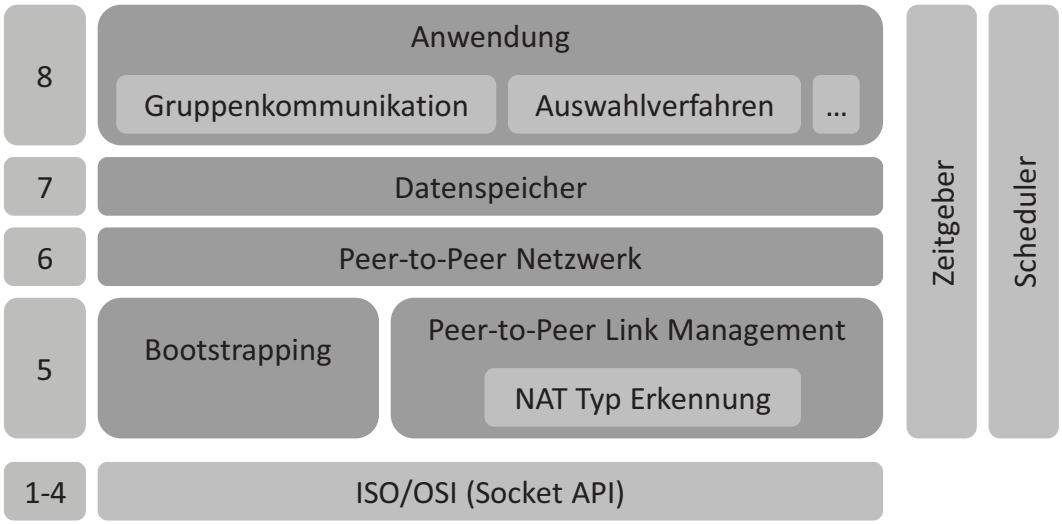


Abbildung 4.2: peers@play Architektur

Struktur. In der peers@play Architektur können in dieser Schicht mehrere Peer-to-Peer Netzwerke existieren und gleichzeitig verwendet werden.

Der größte Unterschied zu existierenden Peer-to-Peer Architekturen ist die Schicht 5, die *Peer-to-Peer Link Schicht* (siehe Abbildung 4.2). Diese Schicht setzt direkt auf der Transportschicht (siehe Abschnitt 2.1.1) auf und besteht aus zwei verschiedenen Komponenten. Die erste Komponente, das Bootstrapping, stellt Funktionen zur Verfügung, um aktive Peers eines bestimmten Peer-to-Peer Netzwerks zu finden. Zu diesem Zweck kann beispielsweise ein IRC-basiertes Verfahren, welches in [46] vorgestellt wurde, verwendet werden. Die zweite Komponente, das *Peer-to-Peer Link Management*, ist verantwortlich für die Verbindungsverwaltung und das Aufbauen direkter Verbindungen zwischen Rechnern, die sich hinter verschiedenen NAT Routern befinden können. Die dafür benötigten NAT Traversal Verfahren sind, wie im vorherigen Kapitel beschrieben, nicht trivial und verursachen Kosten in Bezug auf die Zeit, die benötigt wird um die Verbindung aufzubauen und in Bezug auf Nachrichten, die über den gemeinsamen Kommunikationskanal gesendet werden. Daher stellt das Peer-to-Peer Link Management eine Verbindung, die einmal erstellt wurde, allen aktiven Peer-to-Peer Netzwerken zur Verfügung. Zu diesem Zweck verwenden alle aktiven Peer-to-Peer Netzwerke dieselbe Peer-to-Peer Link Management Komponente.

Alle Komponenten und Schichten innerhalb der peers@play Architektur kommunizieren durch asynchronen Nachrichtenaustausch miteinander. Zusätzlich wird innerhalb der peers@play Architektur ein dedizierter Scheduler und Zeitgeber verwendet.

Diese werden von allen entwickelten Schichten und Komponenten verwendet und ermöglichen es denselben Quelltext für Simulationen und für die Release Software zu verwenden. Der Scheduler kontrolliert hierbei die Ausführung aller Komponenten und Schichten und der Zeitgeber ist verantwortlich für alle zeitgesteuerten Ereignisse, Intervalle und Timeouts.

Im Folgenden werden alle Schichten und Komponenten der peers@play Architektur näher beschrieben.

4.2.1 Schicht 5: Bootstrapping

Die erste Aufgabe eines jeden Peers besteht darin eine Verbindung zu einem anderen aktiven Peer aufzubauen oder zu erkennen, dass er der erste Peer ist. In Simulationen oder in kontrollierten Testsystemen kann das Problem des Bootstrappings einfach gelöst werden, indem IP Adressen von aktiven Peers fest in die Software eingetragen werden oder im lokalen Netzwerk eine Broadcast Nachricht gesendet wird. Wenn allerdings die Applikation unter realen Bedingungen in einem Feldexperiment getestet werden soll, benötigt man komplexere Protokolle. Die Standardtechniken hierfür sind beispielsweise Peer- oder Host-Caches. Zusätzlich zu diesen Standardtechniken kann beispielsweise ein auf dem IRC-Protokoll basierendes Bootstrapping Verfahren [45] eingesetzt werden.

Bootstrapping Verfahren nutzen Protokolle aus der Transportschicht, um mit einem externen Service (z.B. DynDNS, IRC, Suchmaschine, usw.) zu kommunizieren. Über diesen externen Service erhält der Peer eine Liste mit Adressen von aktiven Peers. Mit Hilfe dieser Adressen kann dann eine Verbindung mit einem Peer oder mehreren Peers aufgebaut werden. Da das Bootstrapping direkt auf der Transportschicht aufsetzt, ist es ein Protokoll der 5. Schicht innerhalb der peers@play Architektur.

Das Bootstrapping Verfahren muss bei der Auswahl der aktiven Peers, die es einem anfragenden Peer nennt, sicherstellen, dass diese Peers direkt per TCP oder UDP kontaktiert werden können. Es existiert zwar eine Reihe an NAT Traversal Verfahren, die einen Verbindungsaufbau zwischen zwei Peers hinter verschiedenen NAT Routern ermöglichen, allerdings setzen diese Verfahren voraus, dass beide Peers entweder über das Peer-to-Peer Netzwerk (siehe Abschnitt 4.2.2) oder einen gemeinsamen Kommunikationskanal (z.B. Server) Nachrichten miteinander austauschen können. Ein Peer, der sich erst ins das Peer-to-Peer Netzwerk integrieren möchte, kann aber noch keine Nachrichten über das Peer-to-Peer Netzwerk senden und die meisten Peer-to-Peer Netzwerke verwenden keine Server. Das bedeutet, dass diese NAT

Traversal Mechanismen für einen sich neu integrierenden Peer nicht zur Verfügung stehen.

Nachdem sich ein Peer erfolgreich in das Peer-to-Peer Netzwerk integriert hat, kann das Bootstrapping Protokoll entweder vorübergehend deaktiviert werden oder der Peer wird selbst zum Bootstrapper und hilft anderen Peers sich in das Peer-to-Peer Netzwerk zu integrieren.

4.2.2 Schicht 5: Peer-to-Peer Link Management

Peers kommunizieren entweder direkt per TCP oder UDP miteinander oder indirekt, indem sie Nachrichten über das Peer-to-Peer Netzwerk routen. Routing ist eine Eigenschaft der Peer-to-Peer Netzwerk Schicht, während die direkte Kommunikation von den meisten Architekturen [5] als Teil der Transportschicht gesehen wird. Allerdings werden zusätzliche Probleme, wie NAT Traversal und Punkt-zu-Punkt Sicherheit nicht von der Transportschicht behandelt. Daher kapselt die peers@play Architektur diese Funktionalitäten in der Peer-to-Peer Link Management Komponente, da diese Funktionalitäten von allen Peer-to-Peer Netzwerken benötigt werden.

Aufgrund dessen, dass NAT Traversal ein wesentlicher Bestandteil der 5. Schicht ist, unterscheidet sich das Adressierungsschema von dem der Transportschicht (IP Adresse und Portnummer). Rechner, die sich hinter einem NAT Router befinden verfügen über zwei IP Endpunkte, einen Internen und einen Externen. Befinden sich zwei Rechner im selben externen Netzwerk und verwenden dieselbe externe IP Adresse, wird zuerst versucht eine Verbindung über den internen IP Endpunkt z.B. 192.168.x.x. aufzubauen. Der Grund hierfür ist, dass die meisten NAT Router kein Hairpinning unterstützen (siehe Abschnitt 3.5.3). In diesem Fall wäre eine Kommunikation über den externen IP Endpunkt nicht möglich. Das würde dazu führen, dass keine Verbindung zwischen den beiden Rechnern aufgebaut werden kann, obwohl sich beide Rechner im selben lokalen Netzwerk befinden. Ein weiterer Grund dafür, dass sich das Adressierungsschema unterscheidet, resultiert aus der Tatsache, dass NAT Router unterschiedliche Verhaltensweisen zeigen beziehungsweise unterschiedliche Eigenschaften haben (siehe Abschnitt 3.5.3). Diese Eigenschaften sind relevant um zu entscheiden, welches NAT Traversal Verfahren eingesetzt werden kann bzw. sollte (siehe Abschnitt 3.3.4). Wenn ein Rechner eine Verbindung zu einem anderen Rechner herstellen möchte, so müssen die Eigenschaften der beiden involvierten NAT Router bekannt sein. Um diese Eigenschaften zu bestimmen, wird das in Abschnitt 3.3 vorgestellte MFB Protokoll verwendet. Die Bestimmung dieser

Eigenschaften benötigt eine gewisse Zeit. Daher wird die Bestimmung der Eigenschaften durchgeführt, wenn das Link Management gestartet wird und anschließend werden diese Informationen in einer Schicht 5 Adresse zusammengefasst. Zusammenfassend besteht eine Schicht 5 Adresse aus:

Interne IP Adresse + Portnummer, Externe IP Adresse + Portnummer, Netzwerkumgebung

Die Netzwerkumgebung beinhaltet hierbei alle vom MFB Protokoll bestimmten Eigenschaften und Verhaltensweisen.

Zwei Rechner, die beispielsweise versuchen eine direkte Verbindung mittels Hole Punching herzustellen, müssen ihre Aktionen koordinieren. Aufgrund dessen, dass beide Rechner zu diesem Zeitpunkt noch nicht miteinander verbunden sind, aber beide Rechner sich in die Netzwerkstruktur des Peer-to-Peer Netzwerks integriert haben, kann das Peer-to-Peer Netzwerk verwendet werden, um die Aktionen zu koordinieren. Zu diesem Zweck kann das Link Management die darüber liegende Schicht, das Peer-to-Peer Netzwerk, auffordern, eine Nachricht über das Peer-to-Peer Netzwerk zu routen (siehe Abbildung 4.3).

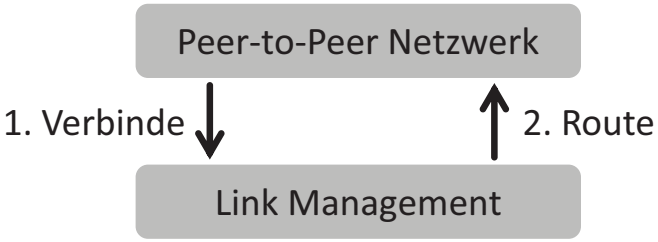


Abbildung 4.3: Link Management und Routen von Nachrichten

Diese Aufforderung ist in Schichtenarchitekturen unüblich, da in der Regel nur die obere Schicht die untere Schicht auffordert etwas auszuführen. Allerdings ermöglicht es NAT Traversal Verfahren ohne Server durchzuführen, indem bereits initialisierte Komponenten verwendet werden. Ohne diese Möglichkeit müsste ein zentraler Server als gemeinsamer Kommunikationskanal zur Verfügung stehen. Alle Peers des Peer-to-Peer Netzwerks müssten mit diesem Server verbunden sein oder es wäre in vielen Fällen nicht möglich eine direkte Verbindung per NAT Traversal herzustellen.

Eine ähnliche Situation tritt auf, wenn ein direkt erreichbarer Peer benötigt wird, der als Vermittler für zwei Peers dienen soll, die keine direkte Verbindung miteinander

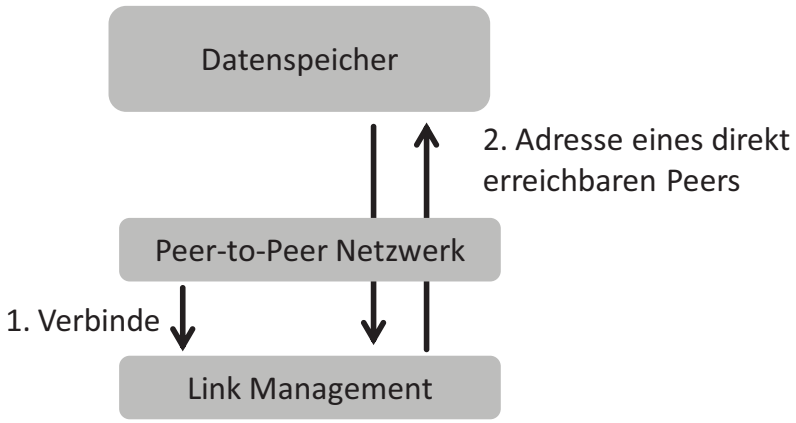


Abbildung 4.4: Link Management und direkt erreichbare Peers

aufbauen können. In der peers@play Architektur wird dieses Problem mit Hilfe des Datenspeichers gelöst (siehe Abbildung 4.4).

Jeder Peer, der direkt erreichbar ist und somit als Vermittler in Frage kommt, speichert seine Adresse unter einem allgemein bekannten Schlüssel im Datenspeicher. Die Peer-to-Peer Link Management Komponente kann somit den Datenspeicher nach Adressen von direkt erreichbaren Peers fragen und eine Verbindung zu einem Vermittler aufbauen.

Die Peer-to-Peer Link Management Komponente implementiert eine Reihe von Sicherheitsmechanismen für Punkt-zu-Punkt Kommunikation, wie beispielsweise die Verschlüsselung von Nachrichten oder die Authentifizierung von anderen Peers mit Hilfe von Zertifikaten. Hierfür verfügt jeder Peer über ein, von einer Zertifizierungsstelle ausgestelltes Zertifikat. Die Peer-to-Peer Netzwerk Schicht kann ebenfalls eine Verschlüsselung von Nachrichten durchführen. Allerdings handelt es sich auf dieser Ebene um eine Ende-zu-Ende und nicht um eine Punkt-zu-Punkt Verschlüsselung, wie sie vom Peer-to-Peer Link Management durchgeführt wird.

Ein weiterer Grund für die Einführung der Peer-to-Peer Link Schicht resultiert aus der Anforderung mehrere aktive Peer-to-Peer Netzwerke zu unterstützen. Wenn mehrere Peer-to-Peer Netzwerke zur selben Zeit ausgeführt werden, können bereits aufgebaute Verbindungen von den Peer-to-Peer Netzwerken gemeinsam genutzt werden. Beispielsweise können Verbindungen, die in einem Peer-to-Peer Netzwerk zur Erhaltung der Struktur aufgebaut wurden, in einem anderen Netzwerk als Abkürzung beim Routing verwendet werden. Hierbei können beide Peer-to-Peer Netzwerke

dieselbe Verbindung verwenden, wodurch die Kosten für den Verbindungsaufbau und die Aufrechterhaltung minimiert werden.

Die Peer-to-Peer Link Management Komponente verwendet eine spezielle Verbindungsverwaltung, die von der klassischen Öffnen/Schließen Semantik von Sockets oder Dateien abweicht. Wenn eine Verbindung einige Zeit nicht mehr verwendet wurde und die Anzahl der aktuell aktiven Verbindungen einen bestimmten Grenzwert überschreitet, fragt die Peer-to-Peer Link Management Komponente alle aktiven Peer-to-Peer Netzwerke nach ihrem Votum. Ein Peer-to-Peer Netzwerk kann hierbei auf der Aufrechterhaltung der Verbindung bestehen, wenn diese Verbindung beispielsweise zur Erhaltung der Netzwerkstruktur dient oder es kann indifferent bezüglich der Verbindung sein. Wenn mindestens ein Peer-to-Peer Netzwerk für die Aufrechterhaltung der Verbindung stimmt, wird die Verbindung nicht geschlossen. Andernfalls kann das Peer-to-Peer Link Management sich dazu entscheiden diese Verbindung zu schließen. In diesem Fall werden alle aktiven Peer-to-Peer Netzwerke darüber informiert, dass die Verbindung geschlossen wurde. Durch diese Verbindungsverwaltung muss kein Peer-to-Peer Netzwerk explizit eine Verbindung schließen, sondern es muss nur zwischen notwendigen Verbindungen und Verbindungen, die verwenden werden, weil sie bestehen, unterscheiden.

Durch die Unterstützung mehrerer aktiver Peer-to-Peer Netzwerke, die dieselbe Peer-to-Peer Link Schicht verwenden, muss das Peer-to-Peer Link Management die Nachrichten aller Peer-to-Peer Netzwerke über die bestehenden TCP/UDP oder RUDP Verbindung multiplexen. Die Peer-to-Peer Link Management Komponente verwendet hierfür einen Nachrichten-Scheduler, um zu entscheiden welche Nachricht direkt gesendet, welche in eine Warteschlange eingereiht und welche eventuell gar nicht gesendet wird. Wenn beispielsweise eine große Datei transferiert werden soll, um einen Teil der Anwendung zu aktualisieren, wird das Peer-to-Peer Link Management diese große Nachricht aufteilen und in die Warteschlange einreihen. Hierdurch können beispielsweise Nachrichten, die zur Aufrechterhaltung der Netzwerkstruktur notwendig sind, verzögert werden. Werden diese Nachrichten zu lange verzögert, kann dies dazu führen, dass andere Peers der Meinung sind, dass dieser Peer nicht mehr aktiv ist. Der Transfer von großen Datenmengen könnte so die Netzwerkstruktur gefährden. Diesem Problem kann mit dynamischen Grenzen für Zeitüberschreitungen entgegengewirkt werden. Allerdings betrifft dieses Problem auch alle Nachrichten, die mit der Interaktivität der Anwendung zu tun haben, so dass die Anwendung für den Nutzer unbenutzbar werden kann. Daher verwendet die Peer-to-Peer Link Management Komponente ein priorisiertes Scheduling vergleichbar mit dem Algorithmus von DiffServ [59]. Insgesamt werden drei verschiedene Prioritäten verwendet:

- Hoch: Nachrichten für Echtzeit Nutzerinteraktion
- Mittel: Nachrichten zur Erhaltung des Peer-to-Peer Netzwerkes
- Niedrig: Dateitransfer

Der Scheduler sendet zuerst alle Nachrichten mit einer hohen Priorität. Um zu verhindern, dass Nachrichten mit einer mittleren Priorität zu lange zurück gehalten werden und so die Netzwerkstruktur gefährden können, wird die Priorität dieser Nachrichten nach einer gewissen Zeit erhöht. Nachrichten, die eine niedrige Priorität haben, werden nur gesendet, wenn genügend Ressourcen verfügbar sind. Die Priorität dieser Nachrichten wird nie erhöht.

Allerdings kann dies dazu führen, dass obere Schichten, die beispielsweise für den Dateitransfer zuständig sind, versuchen ihre Nachrichten erneut zu schicken, da diese zu lange zurückgehalten wurden und sie somit keine Bestätigung über den Empfang erhalten haben. Um dies zu verhindern, stellt das Peer-to-Peer Link Management den oberen Schichten eine Schnittstelle zur Verfügung mit der die aktuelle Länge der Warteschlange überprüft werden kann. Eine Schicht, die sich korrekt verhält und regelmäßig Nachrichten mit einer niedrigen Priorität sendet, überprüft die Länge der Warteschlange, um so gegebenenfalls seinen Datenfluss anzupassen. Auf den ersten Blick kann dies nach einem Fehler im Schichtendesign aussehen, da die Flusskontrolle eine Aufgabe der 4. bzw. 5. Schicht ist. Allerdings ist es in Peer-to-Peer Dateiaustauschprotokollen üblich die Flusskontrolle in höheren Schichten abzuwickeln. BitTorrent [19] beispielsweise verwendet eine Flusskontrolle um *Free-Rider* zu erkennen und anschließend zu drosseln. Um BitTorrent in den Schichten 6 und 7 zu implementieren benötigen diese Schichten die Möglichkeit den Datenfluss zu kontrollieren, was die Möglichkeit, die Länge der Warteschlange zu bestimmen mit einschließt.

4.2.3 Schicht 6: Peer-to-Peer Netzwerk

Die Peer-to-Peer Netzwerk Schicht ist vergleichbar mit der Peer-to-Peer Basis Schicht der Schichtenarchitektur von Aberer [5]. Die zentrale Aufgabe dieser Schicht ist der Aufbau und die Instandhaltung einer Netzwerkstruktur sowie das Routing von Nachrichten. Den oberen Schichten stellt die Peer-to-Peer Netzwerkschicht eine Schnittstelle zur Verfügung mit der eine Nachricht an jeden aktiven Peer gesendet werden kann. Hierfür muss die Schicht 6 Adresse des Peers bekannt sein (z.B. ein SHA1 Wert). Die Peer-to-Peer Netzwerk Schicht muss eine Reihe Zuordnungen von Schicht

5 zu Schicht 6 Adressen in einer Routing Tabelle speichern. Wenn eine Nachricht zu einem Peer, der sich in der Routing Tabelle befindet gesendet werden soll, hat die Peer-to-Peer Netzwerkschicht die drei folgenden Möglichkeiten diese Nachricht zu senden:

- Routing: routet die Nachricht über bestehende Schicht 5 Verbindungen
- Direkte Verbindung: baut über das Peer-to-Peer Link Management eine direkte Verbindung auf, wenn die Schicht 5 Adresse bekannt ist
- Hybrid: routet die Nachricht sofort und versucht parallel dazu eine direkte Verbindung aufzubauen

Befindet sich der Peer nicht in der Routing Tabelle, so ist die Schicht 5 Adresse des Peers nicht bekannt und die zweite Möglichkeit kann nicht verwendet werden. Welche dieser Möglichkeiten gewählt wird, ist abhängig von der Anzahl, der Größe sowie der Dringlichkeit der zu sendenden Nachrichten.

Wenn eine Anwendung nur einige kleine Nachrichten mit hoher Dringlichkeit versenden möchte, ist es meistens schneller diese Nachrichten über mehrere Peers zu routen, als darauf zu warten, dass eine direkte Verbindung aufgebaut ist. Allerdings verschwendet das Routing die Bandbreite aller auf dem Pfad befindlichen Peers und sollte somit nur für wenige kleine Nachrichten verwendet werden. Sollten mehrere oder größere Nachrichten gesendet werden, so ist eine direkte Verbindung zu bevorzugen. Muss die Anwendung mehrere Nachrichten mit hoher Dringlichkeit senden, sollte die hybride Möglichkeit gewählt werden.

Das Peer-to-Peer Netzwerk kann allerdings nicht vorhersehen, wie viele Daten die Anwendung senden wird und kann auch nicht die Dringlichkeit der Nachrichten bestimmen. Daher müssen die oberen Schichten dem Peer-to-Peer Netzwerk einen Optimierungshinweis geben. Das geschieht in der peers@play Architektur mit Hilfe von zwei Flags. Die oberen Schichten können hierbei angeben, ob sie eine größere Menge an Nachrichten schicken möchten (Bandbreiten-Flag). Zusätzlich kann angeben werden, ob es sich um dringliche Nachrichten handelt (Dringlichkeits-Flag). Basierend auf diesen Flags und dem Wissen über die Schicht 5 Adresse, kann die Peer-to-Peer Netzwerk Schicht die optimale Möglichkeit auswählen.

Einige Peer-to-Peer Netzwerke können die spezielle Verbindungsverwaltung des Peer-to-Peer Link Managements nutzen um ihre Routing Tabelle zu optimieren. Als Beispiel kann das Peer-to-Peer Netzwerk Pastry, wenn es seine Routing Tabelle füllt, jeden Peer aufnehmen, dessen Adresse einen passenden Präfix hat. Hierdurch können Nachrichten schneller geroutet werden, da zu diesen Peers bereits eine Verbindung

besteht und nicht erst bei Bedarf eine Verbindung aufgebaut werden muss. Zusätzlich können auf diese Weise mehr Verbindungen geteilt werden, wodurch weniger Bandbreite für die Instandhaltung und weniger Ressourcen benötigt werden. Mit einer entsprechenden Öffnen/Schließen Semantik wäre dies nicht möglich. Durch die spezielle Verbindungsverwaltung der Peer-to-Peer Link Management Komponente wird die Unterstützung mehrerer aktiver Peer-to-Peer Netzwerke optimiert.

Im Gegensatz zu der Peer-to-Peer Basis Schicht der Architektur von Aberer [5] existiert in der peers@play Architektur keine verbindliche Schnittstelle für die Peer-to-Peer Netzwerkschicht. Hierfür gibt es zwei Gründe. Der erste Grund ist, dass die verschiedenen Peer-to-Peer Netzwerke unterschiedliche Adressierungsarten verwenden. Während das Peer-to-Peer Netzwerk Chord [80] oder Pastry [71] eindimensionale Adressen verwenden, verwendet das Peer-to-Peer Netzwerk CAN [64] n-dimensionale Adressen. Zweitens unterstützen unstrukturierte Peer-to-Peer Netzwerke, wie Gnutella nur das Fluten von Nachrichten und kein Routing. Zusätzlich existieren Peer-to-Peer Datenspeicher, die auf spezielle Eigenschaften von Peer-to-Peer Netzwerken basieren. Beispielsweise verwendet der Peer-to-Peer Datenspeicher des Peer-to-Peer Netzwerks Pastry einen speziellen Replikationsmechanismus, der auf den linken und rechten Nachbarn der von Pastry verwendeten Ringstruktur basiert. Dieses Konzept der linken und rechten Nachbarn gibt es beispielsweise bei CAN nicht. Anstatt eine verbindliche Schnittstelle mit dem kleinsten gemeinsamen Nenner zu definieren, ist es in der peers@play Architektur möglich differenzierte Schnittstellen zu verwenden.

Dennoch lassen sich beispielsweise für eindimensionale strukturierte Peer-to-Peer Netzwerke Schnittstellen definieren, die mindestens das Routen von Nachrichten beinhalten. Abhängig von den oberen Schichten können, wenn diese einfache Schnittstelle ausreicht, verschiedene Peer-to-Peer Netzwerke mit der gleichen Datenspeicherungsschicht verwendet werden.

4.2.4 Schicht 7: Datenspeicher

Die Aufgaben des Peer-to-Peer Datenspeichers können neben dem verteilten Speichern von Daten unter anderem Replikation, Versionierung oder Zugriffsverwaltung umfassen. Analog zur Architektur von Aberer existieren zwei unterschiedliche Schichten für das Peer-to-Peer Netzwerk und den Peer-to-Peer Datenspeicher. Allerdings können einige Funktionen des Datenspeichers stark abhängig vom darunter liegenden Peer-to-Peer Netzwerk sein, so dass einige Datenspeicher nur mit

bestimmten Peer-to-Peer Netzwerken funktionieren. In einigen Peer-to-Peer Systemen werden Daten nicht nur auf dem zuständigen Peer gespeichert, sondern auch auf dem Pfad zu diesem Peer. Hierfür muss der Datenspeicher alle Nachrichten, die von der Peer-to-Peer Netzwerk Schicht weitergeleitet werden mitlesen, um den Cache des Datenspeichers aus Schicht 7 aufzubauen. Ein Beispiel für solch ein System ist *Tapestry* [89].

Trotz dieser Abhängigkeiten sind das Peer-to-Peer Netzwerk und der Datenspeicher auch in der peers@play Architektur in separaten Schichten. Hierdurch können beispielsweise verschiedene Datenspeicher mit demselben Peer-to-Peer Netzwerk verwendet werden oder es kann auch ein Peer-to-Peer Netzwerk ohne Datenspeicher verwendet werden.

4.2.5 Schicht 8: Applikation

Die Applikationsschicht der peers@play Architektur beinhaltet die Applikationslogik sowie die Anbindung an die Visualisierung. Bei der Entwicklung von Peer-to-Peer Applikationen gibt es eine Reihe häufig wiederkehrender Probleme. Allerdings sind diese Probleme nicht allgemein genug, um hierfür eine eigene Schicht innerhalb der peers@play Architektur zu erstellen. Daher werden die Verfahren, die diese Probleme lösen in Komponenten innerhalb der Applikationsschicht gekapselt. Hierdurch wird die Wiederverwendbarkeit erhöht und Entwicklungszeit reduziert.

Ein Beispiel für eine solche Komponente ist die *Gruppenkommunikation*. Mit Hilfe dieser Komponente kann eine Applikation Interessengruppen mit einer beliebigen Anzahl an Peers erstellen. Diese Interessengruppen können dazu verwendet werden eine Nachricht an alle Peers dieser Gruppe zu senden. Zum einen kann diese Funktionalität in Chat Anwendungen verwendet werden, in denen mehrere Nutzer an einem gemeinsamen Chat teilnehmen. Zum anderen kann diese Funktionalität in Applikationen verwendet werden in denen ein Nutzer mehrfach über verschiedene Endgeräte zur selben Zeit angemeldet sein kann. Um die Synchronisation dieser Instanzen zu vereinfachen, können alle Instanzen eines Nutzers eine Interessengruppe bilden. Wenn immer eine Nachricht an diese Interessengruppe gesendet wird, wird sie automatisch an alle Instanzen gesendet.

Ein weiteres häufig auftretendes Problem in verteilten Systemen betrifft *Auswahlverfahren*. Daher wird diese Funktionalität ebenfalls in einer Komponente gekapselt. In manchen Peer-to-Peer Systemen werden bestimmte Peers aufgrund ihrer Eigenschaften, wie hoher Bandbreite oder Verfügbarkeit ausgewählt. Die Auswahlkomponente

erhält eine Reihe Anforderungen und führt ein Auswahlverfahren durch, um einen passenden Peer zu finden.

4.2.6 Scheduler und Zeitgeber

Eine wesentliche Anforderung an die peers@play Architektur ist, dass derselbe Quelltext für Simulationen als auch für die eigentliche Anwendung verwendet werden kann. Hierbei ist es wichtig, dass eine große Anzahl an Peer Instanzen auf einem einzelnen Rechner simuliert werden kann. Daher muss mit Systemressourcen wie Speicher und Threads sorgfältig umgegangen werden. Zu diesem Zweck wurde innerhalb der peers@play Architektur das in Abschnitt 2.11 vorgestellte Programmiermodell Gears4Net eingesetzt.

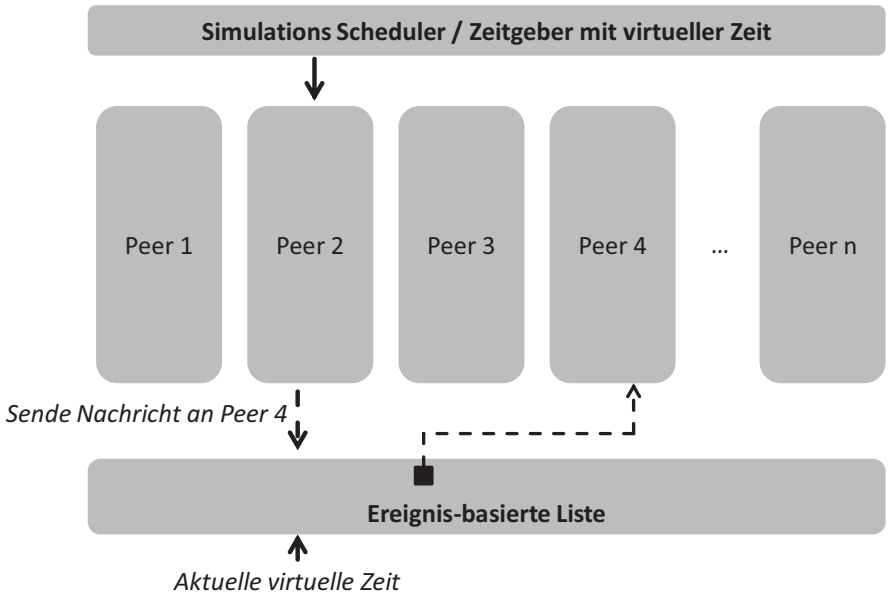


Abbildung 4.5: Diskrete Ereignis-basierte Simulation

Um eine diskrete Ereignis-basierte Simulation einer Anwendung durchzuführen, müssen die eigentliche Ausführung und das zeitliche Verhalten der Anwendung kontrollierbar sein. Zu diesem Zweck wird ein zentraler Scheduler, der die Ausführung kontrolliert und ein zentraler Zeitgeber für das zeitliche Verhalten verwendet (siehe Abbildung 4.5). Der zentrale Scheduler wird von allen Peer Instanzen und somit von allen Gears4Net Protokollen verwendet. Hierdurch kontrolliert der Scheduler die gesamte Ausführung und gewährleistet eine deterministische Ausführung der Anwendung.

In einer Ereignis-basierten Simulation erfolgt der Simulationsfortschritt durch Ereignisse, die zu einer bestimmten virtuellen Zeit ausgelöst werden. Diese Ereignisse treten zu einer bestimmten virtuellen Zeit ein und können wiederum neue Ereignisse in der Zukunft auslösen. Daher verwenden alle Instanzen denselben Zeitgeber, um beispielsweise einen Timeout zu definieren, der dann als Ereignis mit einer bestimmten virtuellen Zeit versehen wird.

Wenn ein Peer einem anderen Peer eine Nachricht senden möchte, so wird in der Simulation kein Netzwerksocket verwendet, sondern es wird ein Ereignis *Nachricht empfangen* in die Ereignisliste eingetragen. Wenn die virtuelle Zeit dieses Ereignisses eintritt, wird dem Peer die Nachricht zugestellt (siehe Abbildung 4.5). Zu diesem Zweck kann bei der Kommunikationskomponente zwischen Sockets und Simulation ausgewählt werden.

Um anstatt einer Ereignis-basierten Simulation die eigentliche Anwendung auszuführen müssen der Scheduler, der Zeitgeber sowie die Komponente zum Versenden von Nachrichten ausgetauscht werden. Hierbei werden zur Kommunikation Netzwerksockets verwendet und anstatt eines Zeitgeber, der mit virtueller Zeit arbeitet, ein Zeitgeber verwendet, der mit realer Zeit arbeitet. Der Scheduler wird bei der Ausführung der eigentlichen Anwendung nur von einem einzelnen Peer verwendet.

Durch den Einsatz von Gears4Net und dem asynchronen Nachrichtenaustausch zwischen den Komponenten und Schichten kann durch den einfachen Austausch der drei Komponenten (Scheduler, Zeitgeber, Komponente zum Versenden von Nachrichten) der selbe Quelltext für Simulationen als auch für die eigentliche Anwendung verwendet werden.

4.3 Verwandte Arbeiten

Im Bereich der Simulationsumgebungen existiert eine Reihe von Simulatoren, die es ermöglichen das unterliegende Netzwerk eines Peer-to-Peer Systems zu simulieren. NS-2 [4] ist ein diskreter Ereignis-basierter Simulator, der eine Simulation auf der Ebene von Netzwerkpaketen ermöglicht. Der Fokus dieser Simulationsumgebung liegt daher auf den ersten vier Schichten des hybriden Referenzmodells. NS-2 stellt keine Funktionalitäten zur Verfügung, um vollständige Peer-to-Peer Anwendungen zu simulieren. Im Gegensatz zu NS-2 können Peer-to-Peer Simulatoren wie PeerSim [54], P2PRealm [48] oder PlanetSim [50] komplette Peer-to-Peer Anwendungen

simulieren. Allerdings wird in beiden Fällen zwischen dem Quelltext für Simulationen und Release Software unterschieden, so dass die Anforderung $R1$ von diesen Ansätzen nicht erfüllt wird.

Im Gegensatz zu den angesprochenen Simulatoren, erlauben es Entwicklungswerkzeuge wie Neko [85], MACEDON [67], RealPeer [35] und OverSim [9] den gleichen Quelltext für Simulationen und die Release Software zu verwenden. Neko [85] zielt auf kleinere verteilte Systeme und ist daher nicht in der Lage eine Vielzahl an Peers zu simulieren, wie es für die Simulation von Peer-to-Peer Netzwerken notwendig ist. MACEDON [67] hingegen bietet eine Infrastruktur, die das Design, die Entwicklung und Evaluation von Peer-to-Peer Netzwerken unterstützt. Der Einsatzzweck von MACEDON ist allerdings eingeschränkt, da MACEDON auf den Einsatz von verteilten Hashtabellen (DHTs) und Multicast auf Applikationsebene ausgerichtet ist. In [35] wurde RealPeer, ein Framework für eine Simulations-basierte Entwicklung von Peer-to-Peer Systemen vorgestellt. RealPeer unterstützt hierbei die Simulation als auch die Entwicklung von Peer-to-Peer Systemen, indem ein Simulationsmodell iterativ zu einer Produktivanwendung transformiert wird. Allerdings bietet RealPeer keine Unterstützung für mehrere aktive Peer-to-Peer Netzwerke. OverSim [9] ist ein skalierendes und flexibles Framework für Peer-to-Peer Netzwerke, das Simulationen und Produktivanwendungen unterstützt. Die in OverSim verwendete Architektur unterstützt Mechanismen für Bootstrapping, Peer-to-Peer Netzwerke und DHTs. Allerdings bietet auch OverSim keine Unterstützung für mehrere zeitgleich aktive Peer-to-Peer Netzwerke.

Zusätzlich zu den bisher betrachteten verwandten Arbeiten existieren eine Vielzahl von Architekturen, wie die von Aberer [5], Dabek [21] und Lua [51]. Diese Arbeiten beschreiben geschichtete Architekturen ohne weitere detaillierte Informationen. Die meisten Architekturen haben separate Schichten für das Peer-to-Peer Netzwerk und den Datenspeicher, aber keine der Architekturen unterstützt die Funktionalitäten der Peer-to-Peer Link Schicht. Zusätzlich bieten diese Architekturen keine Funktionalitäten um denselben Quelltext für Simulationen und Produktivanwendungen zu nutzen.

Peer-to-Peer Frameworks wie JXTA [29], SpoVNet [12], und OPeN [82] sind nicht generisch genug in Bezug auf die unterstützten Anwendungen. Des Weiteren fehlen auch in diesen Frameworks die Funktionalitäten der Peer-to-Peer Link Schicht und die Unterstützung für mehrere aktive Peer-to-Peer Netzwerke.

4.4 Evaluation

In diesem Abschnitt wird die peers@play Architektur bezogen auf Speicherverbrauch und Verarbeitungszeit evaluiert. Alle Schichten, Komponenten und Testfälle, die für die Evaluation verwendet wurden, wurden mit Microsoft .NET C# 4.0 implementiert. Ausgeführt wurde die Evaluation auf einem einzelnen Lenovo ThinkPad T61p mit einem 2.6Ghz Prozessor, 3GB RAM und Windows 7 (32-bit) als Betriebssystem.

4.4.1 Verarbeitungszeit

Im Folgenden wird evaluiert, wie viel zusätzliche Verarbeitungszeit durch die Verwendung der peers@play Architektur und Gears4Net erzeugt wird. Hierfür wird die Umlaufzeit einer Nachricht in verschiedenen Testszenarios ermittelt. Die Nachricht läuft hierbei auf der einen Seite durch alle Schichten nach unten, wenn sie gesendet wird und durch alle Schichten wieder nach oben, wenn sie empfangen wird. Dieser Nachrichtenfluss ist in Abbildung 4.6 dargestellt und gleicht dem der peers@play Architektur.

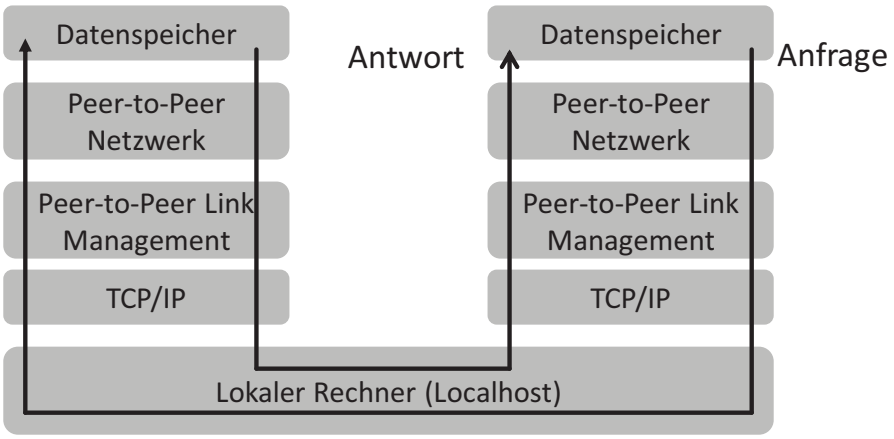


Abbildung 4.6: Nachrichtenfluss einer Speicher-Operation

Für das erste Testszenario wurden zwei Testprogramme implementiert, wodurch die zusätzliche Verarbeitungszeit ermittelt wird, die durch die Aufteilung einer Applikation in verschiedene Schichten und den Einsatz von Gears4Net erzeugt wird. Das Testprogramm *A* sendet alle Nachrichten direkt über einen TCP Socket ohne den Einsatz von Gears4Net und ohne Aufteilung in verschiedene Schichten. Testprogramm *B* hingegen verwendet drei verschiedene Schichten wie in Abbildung 4.6 dargestellt. Jede Schicht leitet die Nachricht hierbei einfach weiter zur nächsten Schicht

ohne weitere Verarbeitung. Die Nachricht wird wie bei A über einen Standard TCP Socket gesendet. Zusätzlich verwendet das Testprogramm B Gears4Net. Subtrahiert man die benötigte Verarbeitungszeit des Testprogramms A von der Verarbeitungszeit des Testprogramms B, so erhält man die zusätzliche Verarbeitungszeit, die durch die Aufteilung in die einzelnen Schichten und den Einsatz von Gears4Net erzeugt wird.

In jeder Messung wurden zwei Nachrichten gesendet, wobei die erste Nachricht 290 Byte groß und die Antwortnachricht 258 Byte groß ist. Mit beiden Testprogrammen wurde 1.000.000 mal die Zeit bis zum Erhalt der Antwortnachricht (Umlaufzeit) gemessen. Die Ergebnisse dieser Messung sind in Tabelle 4.1 zusammengefasst. Die Spalten $Q1$ und $Q3$ geben hierbei die Werte für das untere und obere Quartil an, welche 50% der ermittelten Werte umfassen.

Zusätzlich sind die Ergebnisse dieser Messungen in Abbildung 4.7 grafisch dargestellt. Die durchschnittliche Zeit unter Verwendung des Testprogramms A lag bei $0,10ms$ und für das Testprogramm B bei $0,12ms$. Subtrahiert man die Umlaufzeit von A von der Umlaufzeit von B, so ergibt sich eine zusätzliche Verarbeitungszeit von $0,02ms$. Berücksichtigt man, dass die durchschnittliche Latenz im Internet zwischen $30ms$ und $40ms$ liegt, so ist diese zusätzliche Verarbeitungszeit vernachlässigbar gering.

	Minimum	Maximum	Q1	Q2 (Median)	Q3	Durchschnitt
A	0,098	0,130	0,099	0,102	0,109	0,105
B	0,108	0,147	0,114	0,120	0,126	0,121
C	0,648	0,752	0,656	0,659	0,665	0,663
D	1,325	1,437	1,337	1,343	1,354	1,348

Tabelle 4.1: Umlaufzeiten der einzelnen Tests (in ms)

Im zweiten Testszenario wurde die Umlaufzeit unter der Verwendung des peers@play Frameworks in zwei verschiedenen Varianten C und D gemessen. Hierfür wurde eine *Speicher*-Operation der Datenspeicher Schicht von einem Peer initiiert und vom anderen Peer verarbeitet. Alle Schichten in diesem Szenario sind voll funktionsfähig. Es wird hierbei die Zeit gemessen bis der initiierende Peer die Bestätigung seiner *Speicher*-Anfrage erhalten hat (siehe Abbildung 4.6). Um die Ergebnisse dieser Messung mit der vorherigen vergleichen zu können, sind die Größen der Nachrichten äquivalent zu den Größen der Testprogramme A und B. In Variante C des peers@play Frameworks wurden die Verschlüsselung der Nachrichten sowie die Replikationsme-

chanismen ausgeschaltet. In Variante D hingegen sind alle Funktionen eingeschaltet und alle Nachrichten werden mit AES (128 Bit) verschlüsselt.

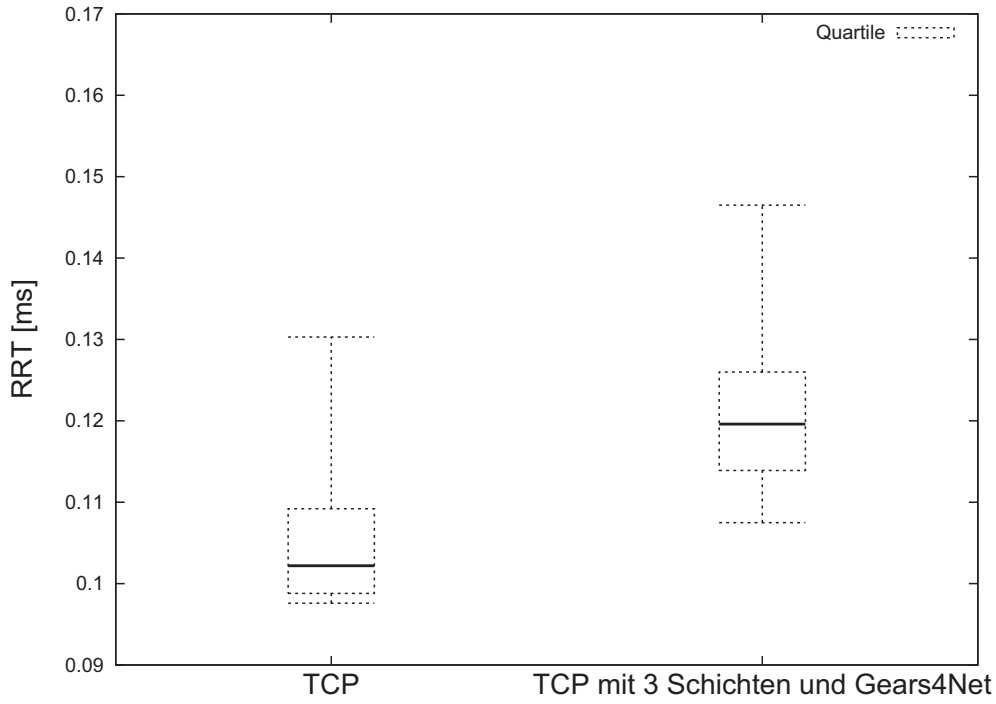


Abbildung 4.7: Umlaufzeit bei Testprogramm A und B

Äquivalent zum ersten Testszenario wurde die Umlaufzeit 1.000.000 mal gemessen. Hierfür wurde das Peer-to-Peer Netzwerk einmalig gestartet und anschließend 1.000.000 mal eine *Speicher*-Operation initiiert. Die durchschnittliche Umlaufzeit in Variante C betrug $0,65ms$ und in Variante D betrug die Umlaufzeit $1,3ms$. Die Ergebnisse dieser Messungen sind in Abbildung 4.8 grafisch dargestellt. Der signifikante Unterschied zwischen den beiden Varianten resultiert aus der Verschlüsselung und Replikation.

In Tabelle 4.1 sind die Ergebnisse aller Tests zusammengefasst. Die Messungen zeigen, dass die durch die Aufteilung in Schichten und den Einsatz von Gears4Net entstehende zusätzliche Verarbeitungszeit sehr gering ist. Der zusätzliche Aufwand für Verschlüsselung und Replikation (siehe C und D) ist um eine Größenordnung höher als die Kosten für die reine TCP Kommunikation (siehe A). Hierdurch wird deutlich welche Bereiche einer Peer-to-Peer Applikation die meiste Verarbeitungszeit benötigen und dass die durch die peers@play Architektur erzielte Flexibilität kaum signifikante zusätzliche Verarbeitungszeit benötigt.

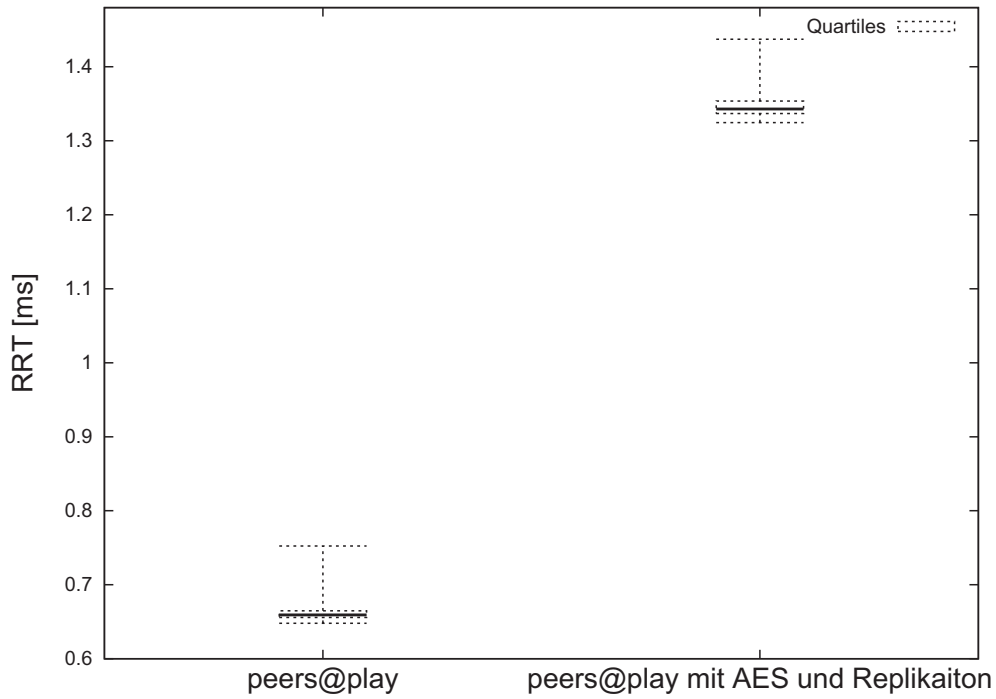


Abbildung 4.8: Umlaufzeit bei Testprogramm C und D

4.4.2 Speicherverbrauch

In diesem Abschnitt wird der Speicherverbrauch der peers@play Architektur evaluiert und somit ermittelt wie viele Peers auf einem einzelnen Rechner simuliert werden können. Zu diesem Zweck wurde ein Peer-to-Peer Netzwerk auf einem Rechner gestartet und der Speicherverbrauch nach jedem neu hinzugefügten Peer ermittelt. Durch den Einsatz von Gears4Net kann die Simulation mit einer festgelegten Anzahl an Threads durchgeführt werden. Im Folgenden wird der physikalische und virtuelle Speicherverbrauch für das in dieser Evaluation verwendete Peer-to-Peer Netzwerk Chord ermittelt. Zusätzlich wird der Speicherverbrauch mit der Microsoft .NET *Speicherbereinigung* ermittelt. Dieser gibt den Speicherverbrauch der Datenstrukturen und Objekte an, die auf dem Heap liegen.

In Abbildung 4.9 ist der Speicherverbrauch in zwei verschiedenen Varianten mit jeweils 1000 Peers grafisch dargestellt. In der ersten Variante wurden zur Kommunikation TCP Sockets und in der zweiten Variante ein In-Prozess Nachrichtenaustausch (IPN) verwendet. Hierbei wird deutlich, dass der Einsatz von TCP deutlich mehr Speicher benötigt als die Variante mit dem In-Prozess Nachrichtenaustausch. Nachdem 1000 Peers dem Peer-to-Peer Netzwerk beigetreten sind, wurden insgesamt 5190 Verbindungen aufgebaut. Der Speicherverbrauch bei TCP war hierbei zwischen 11%

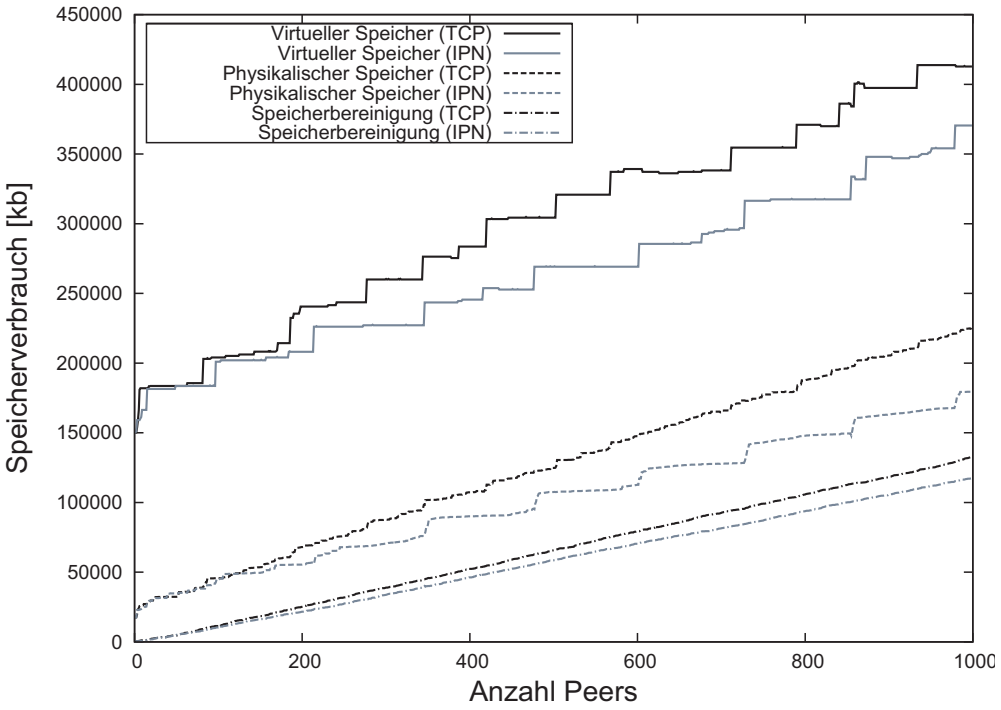


Abbildung 4.9: 1000 verbundene Peers (Chord)

und 25% höher als beim In-Prozess Nachrichtenaustausch. Die Messungen zeigen, dass durch die Verwendung des In-Prozess Nachrichtenaustauschs im Vergleich zu TCP eine größere Anzahl an Peers simuliert werden kann.

Nachdem der unterschiedliche Speicherverbrauch zwischen TCP und dem In-Prozess Nachrichtenaustausch verdeutlicht wurde, zeigt die folgende Messung den Speicherverbrauch für eine größere Anzahl an Peers mit In-Prozess Nachrichtenaustausch. Hierbei wurde der Speicherverbrauch mit 5000 Peers und insgesamt 26914 Verbindungen gemessen. Die Ergebnisse sind in Abbildung 4.10 grafisch dargestellt. Die Simulation von 5000 Peers mit der peers@play Architektur benötigt ca. 800 MB physikalischen Speicher. Die Messung zeigt, dass eine große Anzahl an Peers mit der peers@play Architektur auf nur einem einzelnen Rechner simuliert werden kann.

Die Messungen in Abbildung 4.9 und in Abbildung 4.10 zeigen jeweils einen deutlichen Unterschied zwischen den verschiedenen Messungen. Der virtuelle Speicherverbrauch ist deutlich höher, da die Speicherallokationsverfahren von Microsoft .NET und Windows größere Mengen Speicher anfordern, bevor dieser benötigt wird. Der Unterschied zwischen dem physikalischen Speicherverbrauch und dem, der durch die .NET Speicherbereinigung ermittelt wurde, resultiert daraus, dass der physikalische Speicherverbrauch den vom Stack verwendeten Speicher sowie den Speicherverbrauch von externen Bibliotheken beinhaltet. Zusätzlich beinhaltet der physika-

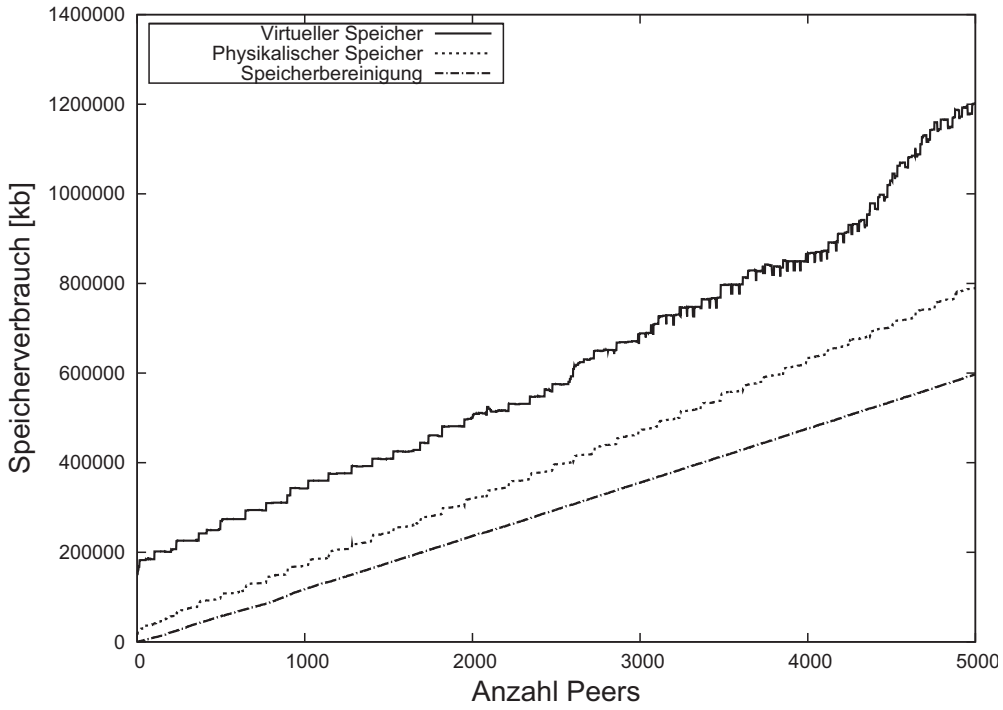


Abbildung 4.10: 5000 verbundene Peers mit In-Prozess Nachrichtenaustausch (Chord)

lische Speicherverbrauch auch den Verlust, der durch die Speicherfragmentierung entsteht. Da der virtuelle Speicherverbrauch schneller wächst als der physikalische Speicherverbrauch, ist die Verwendung eines 64-Bit Rechners sinnvoll, auch wenn der physikalische Speicherverbrauch unter 4 GB liegt.

Zusammenfassend zeigt die Evaluation, dass mit der peers@play Architektur eine große Anzahl an Peers auf einem einzelnen Rechner simuliert werden kann. Durch den Einsatz von Gears4Net und dem In-Prozess Nachrichtenaustausch ist die Größe für den Heap die einzige Ressource, die die Anzahl der zu simulierenden Peers limitiert.

4.5 Zusammenfassung

In diesem Kapitel wurde die peers@play Architektur, die von einer Vielzahl an praktischen Erfahrungen beeinflusst wurde, vorgestellt. Die peers@play Architektur erlaubt es denselben Quelltext für Simulationen als auch für die Produktivanwendung zu verwenden und erfüllt die in Abschnitt 4.2 aufgestellten Anforderungen. Des Weiteren bildet die peers@play Architektur die Grundlage des peers@play Frameworks, welches Bestandteil von zwei Anwendungen ist. Die erste Anwendung ist ein

Prototyp einer vollständig Peer-to-Peer-basierten virtuellen 3D Welt, die auf der CeBit'2010 vorgestellt wurde. Bei der zweiten Anwendung handelt es sich um CrypTool [2], einer E-Learning-Plattform für Kryptographie und Kryptoanalyse. Hier bildet das peers@play Framework die Grundlage für die verteilte Kryptoanalyse.

Durch die Verwendung des Programmiermodells Gears4Net und eines speziellen Schedulers und Zeitgebers ist es möglich denselben Quelltext für Simulationen und Messungen als auch für die Produktivanwendung zu verwenden. Die peers@play Architektur beziehungsweise das Framework wurde hinsichtlich des Speicherverbrauchs und der Verarbeitungszeit evaluiert. Die ermittelten Ergebnisse zeigen, dass die Vorteile, die sich aus der peers@play Architektur ergeben, nur einen geringen Overhead verursachen, so dass die erste Anforderung erfüllt wird. Durch die Aufteilung in Schichten und Komponenten können diese einfach durch verschiedene Implementationen ausgetauscht werden, was die Verwendbarkeit und Austauschbarkeit gewährleistet (R2). Durch die Einführung der Peer-to-Peer Link Schicht ist es möglich mehrere aktive Peer-to-Peer Netzwerke zeitgleich zu unterstützen, so dass R3 erfüllt ist.

Kapitel 5

Verlässliche Datenverteilung

Virtuelle Welten, wie das von Blizzard veröffentlichte World of Warcraft, unterhalten weltweit Millionen von Nutzern. Nutzer spielen dabei die Rolle eines Helden (Avatar) in einer Fantasiewelt, kämpfen gegen Gegner, lösen Rätsel und erfüllen Aufgaben, um stärker zu werden und um neue Ausrüstungsgegenstände zu erhalten. Hierbei dauert es Tage oder Wochen einen Avatar auf den höchsten Level zu bringen. Dabei ist es für einen Nutzer nicht akzeptabel den Fortschritt seines Avatars zu verlieren, weil sein Computer oder die Netzwerkverbindung ausfällt. Insgesamt wird erwartet, dass der Zustand bzw. Fortschritt des Avatars verlässlich gespeichert wird. Ebenso ist es für den Nutzer inakzeptabel, wenn er gegen eine Menge von Gegnern kämpfen musste, um einen bestimmten Ort zu erreichen und er wieder gegen diese Gegner kämpfen muss, weil sein Computer kurz ausgefallen ist. Daher muss der aktuelle Stand der virtuellen Welt verlässlich gespeichert werden. Zudem muss ein Nutzer in der Lage sein den aktuellen Stand abzufragen, nachdem er die virtuelle Welt verlassen hat und dieser wieder neu beitreten möchte.

In Peer-to-Peer-basierten virtuellen Welten existieren keine Server, die zur Speicherung der Daten genutzt werden könnten. Daher müssen die Daten verteilt auf den Peers gespeichert werden. Die Menge der aktiven Peers ist allerdings nicht konstant, sondern Peers betreten und verlassen das Peer-to-Peer Netzwerk ständig und dabei kann man nicht davon ausgehen, dass das Verlassen immer regelkonform durchgeführt wird. Daher ist die einzige Möglichkeit Daten verlässlich zu speichern, diese redundant im Peer-to-Peer Netzwerk zu speichern.

Zusätzlich zu der Tatsache, dass Peers nicht zuverlässig sind, kann nicht davon ausgegangen werden, dass alle Nutzer der virtuellen Welt ehrlich sind (böswillige Peers). Daher muss verhindert werden, dass Nutzer den Zustand der virtuellen Welt zu ihrem Vorteil ändern.

In diesem Kapitel wird ein Replikationsverfahren vorgestellt, das Daten in einem Positions-basierten Peer-to-Peer Netzwerk an verschiedene Positionen im Peer-to-Peer Netzwerk repliziert [36]. Darauf aufbauend wird ein Routingverfahren beschrieben, das es erlaubt diese Replikate auf disjunkten Routingpfaden zu erreichen. Welchen Einfluss böswillige Peers auf die Verfügbarkeit der Replikate haben, wird in Abschnitt 5.4 evaluiert. Die Ergebnisse dieses Kapitels werden in Abschnitt 5.5 zusammengefasst.

5.1 Systemmodell

Im Folgenden wird das Systemmodell, das in diesem Kapitel verwendet wird, näher betrachtet. Es wird angenommen, dass alle Nutzer ihren mit dem Internet verbundenen Computer nutzen, um dem Peer-to-Peer Netzwerk beizutreten. Hierbei wird davon ausgegangen, dass die Rechenleistung und Speicherkapazitäten der Computer höher sind als die, die jeder für die Berechnung der virtuellen Welt benötigt. Dadurch ist es möglich die nicht benötigten Ressourcen für redundante Berechnungen und Datenspeicherung zu verwenden. Weiterhin wird davon ausgegangen, dass die verfügbare Bandbreite und dabei insbesondere der Upstream ein Flaschenhals darstellen kann, da die meisten Nutzer über Breitband ADSL mit dem Internet verbunden sind.

Peers können über eine verlässliche Verbindung, wie sie durch TCP gewährleistet wird, miteinander kommunizieren. Hierbei sind die Peers in der Lage die Verbindung mit Hilfe von NAT Traversal Mechanismen aufzubauen (siehe Abschnitt 2.6). Allerdings können diese eine gewisse Zeit benötigen um eine Verbindung aufzubauen.

Peers betreten und verlassen das Peer-to-Peer Netzwerk zu beliebigen Zeitpunkten. Hierbei kann nicht davon ausgegangen werden, dass Peers das Peer-to-Peer Netzwerk regelkonform verlassen, da Peers ohne Vorankündigung ausfallen können. Allerdings wird davon ausgegangen, dass Peers, die dem Peer-to-Peer Netzwerk der virtuellen Welt beitreten, auch einige Stunden in diesem verweilen [83].

Der Provider der virtuellen Welt ist nicht in der Lage eine große Menge an eigenen Peers im Peer-to-Peer Netzwerk zu betreiben. Dadurch kann nicht sichergestellt werden, dass alle Teilnehmer einem gegebenen Protokoll folgen, was dazu führt, dass Peers nicht vertrauenswürdig sind. Es kann dementsprechend Peers geben, die von Betrügern kontrolliert werden und versuchen den aktuellen Zustand der virtuellen zu verändern. Des Weiteren können diese Betrüger auch zusammenarbeiten und

sich unter einander koordinieren, um einen Betrug durchzuführen. Dabei können sie byzantinisches Verhalten zeigen und sich zu jeder Zeit an das Protokoll halten oder dagegen verstoßen. Es wird allerdings davon ausgegangen, dass die Mehrheit der Nutzer ehrlich ist und sich an das gegebene Protokoll hält. Sollte die Mehrheit in der virtuellen Welt betrügen und sich somit nicht an das Protokoll halten, macht das Betrügen kaum noch Sinn, da man dadurch kaum noch einen Vorteil gegenüber den anderen Betrügern hat.

Aufgrund der Tatsache, dass nicht davon ausgegangen wird, dass vertrauenswürdige Peers im Peer-to-Peer Netzwerk sind, muss der Betreiber der virtuellen Welt eine Zertifizierungsstelle zur Verfügung stellen. Mit Hilfe dieser Zertifizierungsstelle können eindeutige Identitäten (Zertifikate) [76] für jeden Nutzer und damit jeden Peer erstellt werden. Durch diese Identitäten kann man den Zutritt zu der virtuellen Welt kontrollieren, indem nur Nutzer mit einer gültigen Identität der virtuellen Welt beitreten können. Sollte ein Nutzer durch böswilliges Verhalten auffallen, kann seine Identität widerrufen werden, so dass er nicht mehr der virtuellen Welt beitreten kann. Damit ein böswilliger Nutzer sich nicht einfach eine neue Identität erstellen lässt, muss das Erstellen dieser Identität in irgendeiner Weise aufwändig für den Nutzer sein. Das kann beispielsweise durch die Bindung an die echte Identität oder eine Gebühr geschehen. Dadurch ist es dann auch nur sehr schwer möglich sich sehr viele Identitäten zu erzeugen um beispielsweise einen sogenannten *Sybil Angriff* [24] durchzuführen. Die erzeugten virtuellen Identitäten können auch zusätzliche Informationen, wie beispielsweise eine Position innerhalb der virtuellen Welt beinhalten. Nutzer der virtuellen Welt können die Identitäten anderer Nutzer mit Hilfe des öffentlichen Schlüssels der Zertifizierungsstelle überprüfen.

5.2 VoroStore

In diesem Abschnitt wird *VoroStore* vorgestellt. VoroStore ist ein Positions-basiertes Peer-to-Peer Netzwerk, das es erlaubt über disjunkte Routingpfade zu verschiedenen Positionen im Peer-to-Peer Netzwerk zu routen. Zusätzlich umfasst VoroStore ein Replikationsverfahren, das Daten im Peer-to-Peer Netzwerk so platziert, dass diese über disjunkte Routingpfade erreicht werden können.

VoroStore umfasst konzeptionell die 6. und 7. Schicht der im vorherigen Kapitel vorgestellten peers@play Architektur (siehe Abbildung 5.1). Das Peer-to-Peer Netzwerk inklusive dem Routingverfahren ist Teil der Peer-to-Peer Netzwerk Schicht, wohingegen das Replikationsverfahren Bestandteil der Datenspeicher Schicht ist. Obwohl

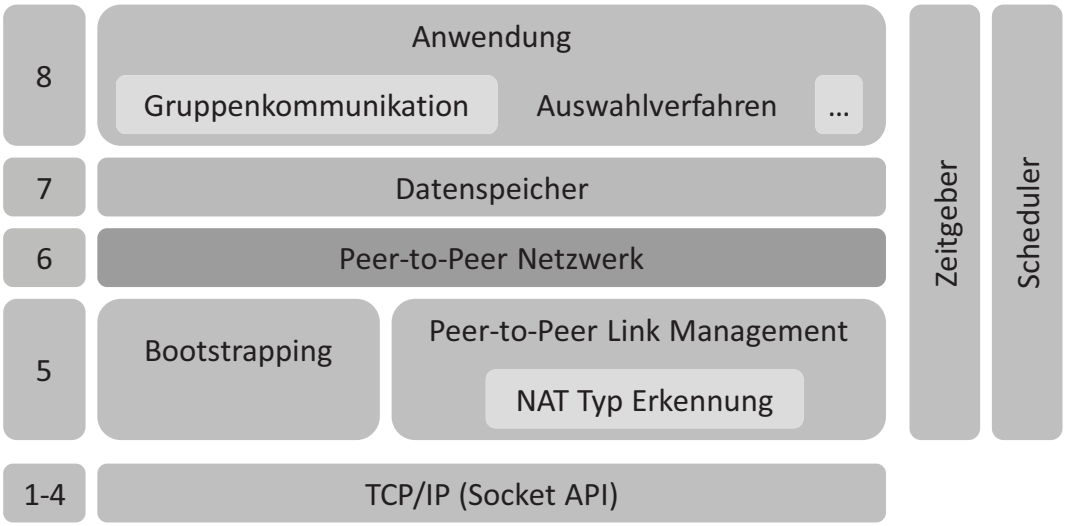


Abbildung 5.1: VoroStore innerhalb der peers@play Architektur

VoroStore mit der Replikation Verfahren aus der Datenspeicher Schicht umfasst, ist VoroStore kein vollständiger Datenspeicher. VoroStore beschreibt wo Daten gespeichert werden und stellt für diese Daten disjunkte Routingpfade zur Verfügung. Aufgaben bezüglich der Konsistenz der Replikate, der Zugriffskontrolle oder ähnliches sind nicht Bestandteil von VoroStore.

Im Folgenden werden die grundlegenden Konzepte von VoroStore näher betrachtet. Das erste Konzept umfasst hierbei Redundanz. Die Redundanz der Daten ist notwendig, um mit der Unzuverlässigkeit und der Nichtvertrauenswürdigkeit der Peers umgehen zu können. VoroStore verwendet zwei verschiedene Arten von Redundanz.

Zum einen wird der allgemeine Ansatz verwendet und die gespeicherten Daten werden auf den Peers repliziert, deren Position nahe an der Position der Daten liegt. Das sind bei VoroStore die Voronoinachbarn. Die Replikate werden hierbei nicht separat adressiert und dienen dazu mit der Unzuverlässigkeit der Peers umzugehen. Damit Peers sich nicht aussuchen können, welche Daten sie verwalten, bekommt jeder Peer eine feste Position zugewiesen. Dafür wird in jeder Identität (Zertifikat) eines Nutzers eine bestimmte Position fest eingetragen. Diese Position bestimmt dann für welche Daten der Peer innerhalb des Peer-to-Peer Netzwerks verantwortlich ist.

Beim zweiten Ansatz wird ein Replikationsschema verwendet, das es erlaubt auf jedes Replikat individuell zuzugreifen (siehe Abschnitt 5.2.1). Hierfür verfügt jedes Replikat über eine eigene Adresse beziehungsweise Position. Um auf diese Replikate

zuzugreifen muss für jedes Replikat eine eigene Anfrage gesendet werden. Die Routingpfade zu diesen Replikaten sind bei VoroStore disjunkt, wie in Abschnitt 5.2.2 gezeigt wird. Im Vergleich zum ersten Replikationsschema dient dieses Replikationsschema dazu den Einfluss von böswilligen Peers zu reduzieren. Ein Angreifer bräuchte für einen erfolgreichen Angriff auf mindestens der Hälfte aller Pfade einen Peer, um eine Anfrage oder das Ergebnis einer Anfrage beeinflussen zu können. Durch die Wahl eines bestimmten Replikationsfaktors k kombiniert mit k disjunkten Routingpfaden, kann die Wahrscheinlichkeit eines Angriffs beeinflusst werden.

In VoroStore werden Voronoi-Diagramme [8] (siehe Abschnitt 2.8) verwendet, um die Fläche der virtuellen Welt in eine Menge von Zellen einzuteilen. Diese Fläche ist hierbei äquivalent zum verwendeten Adressraums des Peer-to-Peer Netzwerks. Jeder Peer ist dabei für eine bestimmte Voronoizelle verantwortlich vergleichbar mit Ansätzen wie [15, 40, 65]. Jeder Peer hat eine feste Position innerhalb der virtuellen Welt und ist dadurch verantwortlich für alle Objekte, die am nächsten an seiner Position sind und somit in seiner Voronoizelle liegen. Ein Peer ist dabei mit seinen direkten Voronoinachbarn verbunden, so dass die grundlegende Struktur des Peer-to-Peer Netzwerks einen Delaunaygraphen (siehe Abschnitt 2.9) bildet. Um die Struktur aufrecht zuhalten verwendet VoroStore den in Abschnitt 2.10 beschriebenen selbststabilisierenden Algorithmus zur Berechnung eines Delaunaygraphen.

Im Gegensatz zu anderen Ansätzen [15, 40, 65] ist die Position eines Peers innerhalb des Peer-to-Peer Netzwerks durch die Zertifizierungsstelle fest zugewiesen. Diese Position ist im Zertifikat des Nutzers eingetragen. Ein Peer kann durch das Verlassen und neu Betreten der virtuellen Welt seine Position und damit seine Zuständigkeit nicht ändern.

Die Position innerhalb des VoroStore Netzwerks ist nicht von der aktuellen Position des Nutzers innerhalb der virtuellen Welt abhängig. Daher muss die Struktur des Peer-to-Peer Netzwerks nur dann angepasst werden, wenn neue Peers hinzukommen oder Peers das Peer-to-Peer Netzwerk verlassen. Zusätzlich zum Peer-to-Peer Netzwerk, das für VoroStore verwendet wird, existiert noch ein zweites dynamisches Peer-to-Peer Netzwerk, das verwendet wird, um die aktuellen Positionen der Nutzer innerhalb der virtuellen Welt auszutauschen [43]. Dieses Netzwerk ist nicht Bestandteil dieser Arbeit und wird daher im Folgenden nicht weiter betrachtet.

Aufgrund der Entscheidung, dass die dynamische Position des Nutzers eine andere ist, als die feste Position innerhalb des Netzwerkes von VoroStore, muss gewährleistet werden, dass Daten, die weit entfernt gespeichert sind, effizient abgefragt werden

können. Zu diesem Zweck wurde das Schema der Routing Tabelle von Pastry [71] erweitert, so dass es in einem zweidimensionalen Raum verwendet werden kann (siehe Abschnitt 5.2.2). Dadurch erreicht VoroStore eine logarithmische Routingkomplexität.

5.2.1 Replikatsplatzierung

Die Positionen der Replikate werden basierend auf der originalen Position des zu speichernden Objekts berechnet. Hierbei haben alle Replikate den gleichen Abstand in jeder Dimension zueinander. Der Abstand der Replikate ist abhängig von der Anzahl der Replikate in jeder Dimension sowie der Größe des Adressraums in der jeweiligen Dimension. Die Größe des Adressraums wird hierbei mit der Breite w und der Höhe h angegeben und der Adressraum wird in k gleich große Segmente unterteilt.

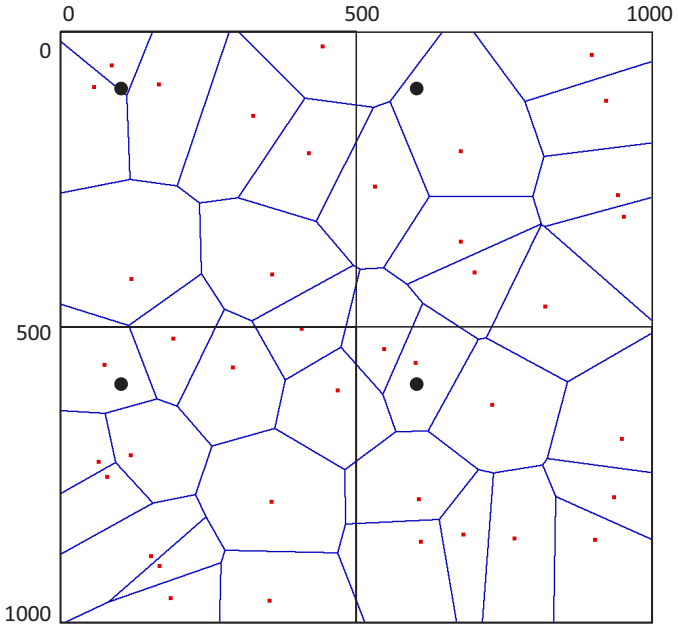


Abbildung 5.2: Replikatsplatzierung

Um die Größe eines jeden Segments zu bestimmen werden zwei Variablen k_x und k_y gewählt, die den Replikationsfaktor pro Dimension festlegen. Hierbei müssen k_x und k_y so gewählt werden, dass sie die Gleichung $k = k_x \cdot k_y$ erfüllen. Die Segmente werden wie folgt als eine Menge von Punkten definiert:

$$S_{i,j} = \{(x,y)|i \cdot \frac{w}{k_x} \leq x < (i+1) \cdot \frac{w}{k_x} \\ j \cdot \frac{h}{k_y} \leq y < (j+1) \cdot \frac{h}{k_y}\}$$

mit $0 \leq i < k_x$ und $0 \leq j < k_y$.

Wird beispielsweise ein Replikationsfaktor $k = 4$ mit $k_x = 2$ und $k_y = 2$ verwendet und der Adressraum ist mit $w = 1000$ und $h = 1000$ definiert, so wird das rechte obere Segment bestimmt durch:

$$S_{1,0} = \{(x,y)|500 \leq x < 1000, 0 \leq y < 500\}$$

Die Positionen für die Replikate werden hierbei so berechnet, dass sich jedes Replikat in einem anderen Segment befindet. Allerdings haben alle Replikate innerhalb eines Segments die gleiche Position relativ zu den Grenzen eines Segments, wie in Abbildung 5.2 dargestellt ist. Für ein Objekt, das die Position $p_{0,0} = (x,y)$ hat, werden die Adressen der Replikate $p_{i,j}$ wie folgt berechnet:

$$p_{i,j} = (x + (i \cdot \tfrac{w}{k_x}) \bmod w, y + (j \cdot \tfrac{h}{k_y}) \bmod h)$$

Wenn beispielsweise die eigentlichen Daten an der Position $p_{0,0} = (100, 100)$ gespeichert werden, so werden die Replikate an den Positionen $(100, 600)$, $(600, 100)$ und $(600, 600)$ gespeichert.

Jeder Peer ist in der Lage die Positionen der Replikate zu berechnen. Dadurch kann jeder Peer eine Anfrage an jedes Replikat senden umso alle Replikate zu erhalten. Basierend auf den erhalten Antworten kann der Peer dann einen Mehrheitsentscheid durchführen. Um den Einfluss eines böswilligen Peers einzugrenzen, müssen alle Anfragen auf disjunkten Pfaden zu den Replikaten gelangen. Wenn dies nicht der Fall wäre, könnte ein böswilliger Peer einfach alle Anfragen verwerfen oder einen fehlerhaften Zustand zurück senden, um so eine Mehrheit für den falschen Zustand zu erhalten. Im folgenden Abschnitt wird ein Routingverfahren vorgestellt, das es erlaubt auf disjunkten Pfaden zu den Replikaten zu routen.

5.2.2 Routing

Das Routingverfahren von VoroStore basiert auf dem Präfix Routing von Pastry [71] und wurde für den Einsatz in einem zweidimensionalen Adressraum erweitert. Vergleichbar mit Pastry verwaltet jeder Peer eine Routing Tabelle und eine Nachbarschaftsliste. Die Nachbarschaftsliste beinhaltet hierbei alle Voronoinachbarn eines Peers. Voronoinachbarn sind hierbei alle Peers mit denen ein Peer verbunden sein muss, um einen Delaunaygraphen zu bilden. Im Gegensatz zu Pastry ist die Größe der Nachbarschaftsliste nicht begrenzt, da die Anzahl der Voronoinachbarn nicht begrenzt ist. Je größer die Nachbarschaftsliste ist, desto mehr Verbindungen müssen verwaltet und aufrecht gehalten werden. Obwohl die Nachbarschaftsliste nicht begrenzt ist, liegt die durchschnittliche Anzahl an Voronoinachbarn, wie in Abschnitt 5.4 gezeigt wird, bei 6 Nachbarn. Die Nachbarschaftsliste garantiert, dass eine Nachricht schlussendlich beim verantwortlichen Peer ankommt.

Im Gegensatz zur Nachbarschaftsliste hat die Routing Tabelle eine feste Größe und stellt Abkürzungen im Peer-to-Peer Netzwerk zur Verfügung um die Anzahl der Routing Schritte zu reduzieren. Zusätzlich garantiert die Routing Tabelle, dass die Pfade zu den Replikaten disjunkt sind. Wenn eine Nachricht geroutet wird, wird die Nachricht in jedem Schritt an den Peer weitergeleitet, der den größten gemeinsamen Präfix mit der Zielposition der Nachricht in beiden Dimensionen hat. Existieren zwei Peers in der Routing Tabelle, wobei einer einen größeren gemeinsamen Präfix in der x-Koordinate und der andere Peer einen größeren gemeinsamen Präfix in der y-Koordinate hat, so wird der Peer verwendet, der den größeren gemeinsamen Präfix in der x-Koordinate hat. Allerdings hätte an dieser Stelle auch der Peer mit dem größeren gemeinsamen Präfix in der y-Koordinate verwendet werden können. Sollte in der Routing Tabelle kein Peer mit einem größeren Präfix existieren, dann befindet sich in der Nachbarschaftsliste ein Peer, der näher an der Zielposition ist oder der Peer selbst ist zuständig für diese Nachricht. Hierdurch ist sichergestellt, dass das Routingverfahren immer konvergiert.

Im Vergleich zu Pastry verwendet VoroStore keinen eindimensionalen Adressraum, sondern einen zweidimensionalen Positionsraum. Daher verwendet VoroStore anstatt der zweidimensionalen Routing Tabelle von Pastry einen dreidimensionalen Routing Quader (siehe Abbildung 5.3). Der Routing Quader besteht aus einer Menge von l verschiedenen Schichten, wobei l die Anzahl der Stellen des Adressraums ist. Die Anzahl der Schichten ist hierbei gleich der Anzahl der Zeilen der Routing Tabelle von Pastry. Jede Schicht umfasst hierbei einen speziellen Bereich des Adressraums wie in Abbildung 5.3 dargestellt.

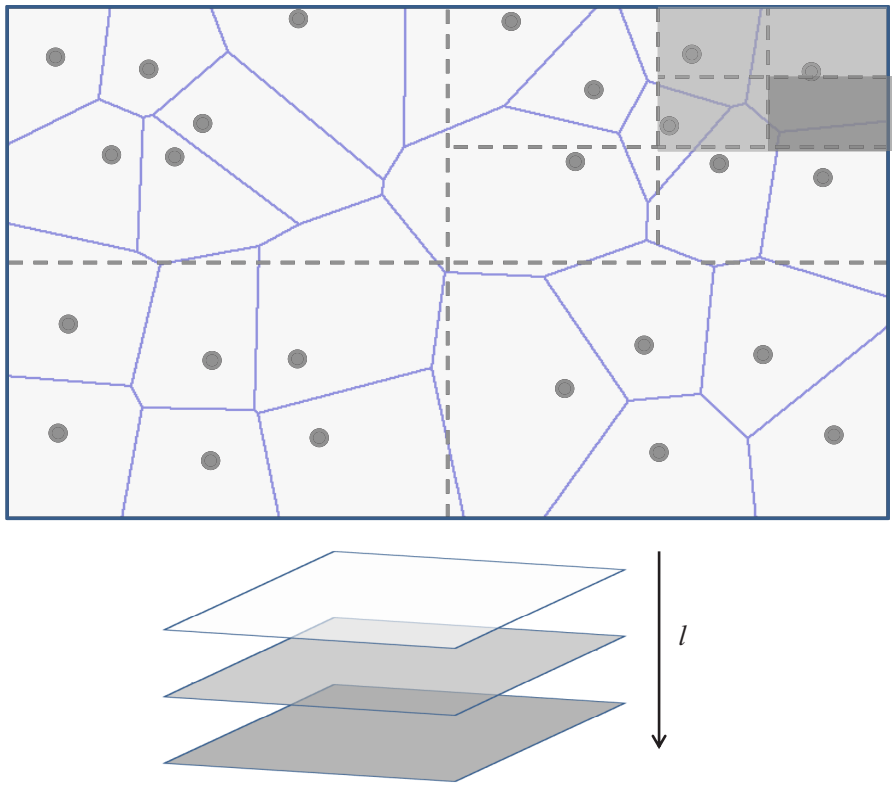


Abbildung 5.3: Routing Quader

Jede Schicht beinhaltet eine Tabelle, die aus b_x Spalten und b_y Zeilen besteht, wobei b_x und b_y durch die Basis des verwendeten Zahlensystems bestimmt werden. In Pastry beinhaltet die oberste Zeile der Routing Tabelle x Einträge, wobei x die Basis des verwendeten Zahlensystems ist. Bei jedem Eintrag muss die führende Zahl der Pastry ID des eingetragenen Peers gleich der Spaltennummer des Eintrags sein. In der Spalte 0 befindet sich ein Peer dessen Pastry ID mit einer 0 beginnt und in Spalte 1 ein Peer dessen Pastry ID mit einer 1 beginnt. Der Routing Quader ist nach demselben Prinzip aufgebaut, allerdings erweitert auf zwei Dimensionen. Die Tabelle der obersten Schicht enthält für den Eintrag in Spalte 0 und Zeile 0 einen Peer dessen Position mit einer 0 in der x-Koordinate und 0 in der y-Koordinate beginnt. In Spalte 0 und Zeile 1 der obersten Tabelle befindet sich ein Peer dessen erste Stelle der x-Koordinate eine 0 und an der ersten Stelle der y-Koordinate eine 1 aufweist.

Um zu entscheiden, an welchen Peer eine Nachricht weitergeleitet werden soll, wird in den Schichten des Routing Quaders nach dem Peer gesucht, der den größten gemeinsamen Präfix in der x- und der y-Koordinate mit der Zielposition hat. Wenn es in den Schichten keinen weiteren Peer gibt, der einen größeren gemeinsamen Präfix

hat, wird überprüft ob in der Nachbarschaftsliste ein Peer existiert, der näher an der Zielposition liegt. Bei diesem Routing Verfahren handelt es sich um einen sogenannten *gierigen Algorithmus*, der schlussendlich einen verantwortlichen Peer findet. Aufgrund der Tatsache, dass bei jedem Routing Schritt der gemeinsame Präfix um eine Stelle wächst, weist das Routing Verfahren eine logarithmische Komplexität auf.

In VoroStore wird eine Nachricht rekursiv weitergeleitet. Das bedeutet, dass wenn eine Nachricht bei einem Peer ankommt, der für diese Nachricht nicht verantwortlich ist, wird der Peer die Nachricht an einen Peer weiterleiten, der näher an der Zielposition ist. Der Absender kennt dementsprechend nicht den Pfad, den die Nachricht nimmt. Wenn eine Nachricht unterwegs verloren geht, kann der Absender nicht entscheiden an welcher Stelle die Nachricht verloren gegangen ist.

Um disjunkte Pfade zu allen Replikaten garantieren zu können, müssen die Basis für b_x der x Dimension und die Basis für b_y der y-Dimension gleich den Faktoren k_x und k_y gewählt werden. Jedes Segment $S_{i,j}$ beinhaltet nur Peers deren Positionen mit i in der x-Koordinate und j in der y-Koordinate beginnen. Die oberste Schicht des Routing Quaders eines jeden Peers beinhaltet daher Peers deren Positionen mit einer Kombination aus i und j beginnen. Das bedeutet, dass jeder Peer mindestens einen anderen Peer aus jedem Segment kennt. Dadurch kann jeder Peer eine Nachricht direkt in das Segment schicken, in dem sich ein Replikat befindet (siehe Abbildung 5.6).

Wenn eine Nachricht einmal in einem Segment ist, wird die Nachricht dieses Segment nicht wieder verlassen, da eine Nachricht wenn sie weitergeleitet wird, immer nur an einen Peer weitergeleitet wird, der einen größeren und niemals einen kleineren gemeinsamen Präfix aufweist. Die einzige Ausnahme in der eine Nachricht ein Segment wieder verlassen kann ist, wenn Peers aus der Nachbarschaftsliste verwendet werden, um die Nachricht weiterzuleiten. Allerdings sollte die Nachbarschaftsliste nur verwendet werden, wenn die Nachricht bereits im Zielgebiet ist. In diesem Fall wären die Routing Pfade trotzdem disjunkt.

5.2.3 Zielpositionen des Routing Quaders

Der Erfolg des disjunkten Pfadroutings ist abhängig von der sicheren Instandhaltung des Routing Quaders. Wenn böswillige Peers in der Lage sind, sich selbst in die Routing Quader von ehrlichen Peers zu platzieren und somit häufiger in Routing Quadern auftreten als andere Peers, dann werden mehr Nachrichten über böswillige

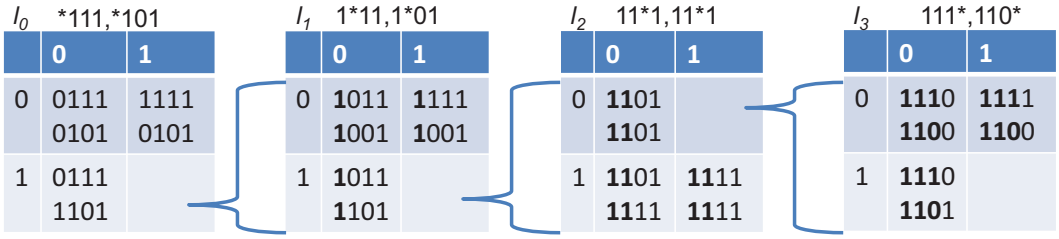


Abbildung 5.4: Routing Quader mit den Schichten $l_0 - l_3$ eines Peers mit der Position (1111,1101)

Peers geleitet als über ehrliche Peers. Dadurch wird die Wahrscheinlichkeit erhöht, dass böswillige Peers eine Mehrheit in den k disjunkten Pfaden erhalten. Deshalb ist es notwendig darauf zu achten, welche Peers in den Routing Quader aufgenommen werden.

Um zu entscheiden, welche Peers in den Routing Quader aufgenommen werden, werden die einzelnen Zellen des Routing Quaders mit Zielpositionen versehen. Die Zielpositionen sind für jeden Routing Quader anders, da sie auf der Position des jeweiligen Peers basieren. Nur Peers, die eine bestimmte Position haben, werden in den Routing Quader aufgenommen. Diese Zielpositionen sind vergleichbar mit den Einschränkungen für die Routing Tabelle von Pastry, die in [18] vorgestellt wurden.

Die Zielposition für die Zelle e in der Schicht i des Routing Quaders eines Peers N wird wie folgt berechnet. Die Ziffern von jeder Koordinate von e müssen gleich den ersten $i-1$ Ziffern der Koordinaten des Peers N sein. Die Ziffer an der Position i ist wie bei den Bedingungen von Pastry abhängig von der Zeile r und der Spalte c. Die verbleibenden Stellen beider Koordinaten für die Zielposition von e müssen gleich den verbleibenden Ziffern der Koordinaten von N sein.

In Abbildung 5.4 sind die einzelnen Tabellen des Routing Quaders für den Peer mit der Position (1111,1101) dargestellt. Die Position in diesem Beispiel ist maximal 4 Ziffern lang (l=4) und verwendet einen binären Zahlenraum (b=2). Die dazugehörige grafische Darstellung der Zielpositionen ist in Abbildung 5.5 dargestellt. Der blaue Punkt gibt hierbei die Position des Peers mit der Position (1111,1101) an. Die roten Punkte sind die Zielpositionen der einzelnen Schichten des Routing Quaders.

Allerdings existiert nicht für jede Zielposition ein Peer, der exakt diese Position hat. Daher werden die Peers verwendet, die am nächsten an der definierten Position liegen. Böswillige Peers könnten jetzt allerdings einfach behaupten, dass sie am nächsten an dieser Position liegen. Damit ein Peer abschätzen kann, ob ein Peer

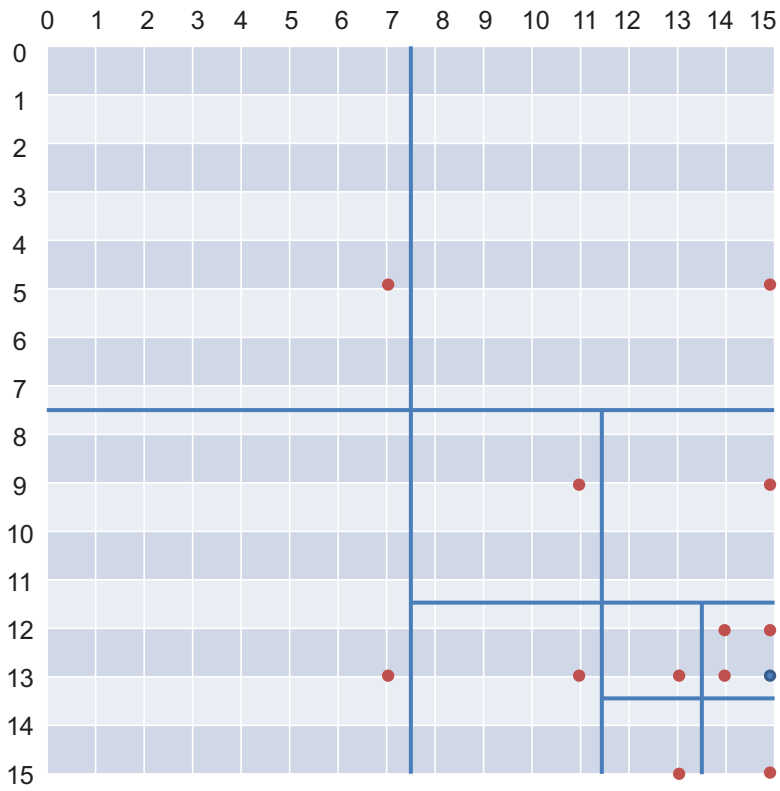


Abbildung 5.5: Zielpositionen der Schichten eines Peers mit der Position (1111,1101)

für eine bestimmte Position verantwortlich sein kann, wird eine maximale Distanz verwendet, die angibt die weit ein Peer von der Zielposition entfernt sein sollte. Die maximale Distanz ist definiert als der durchschnittliche Abstand zwischen 2 Peers. Dieser durchschnittliche Abstand ist abhängig von der Peerdichte im Peer-to-Peer Netzwerk. Um die Peerdichte im Netzwerk abzuschätzen, kann ein Peer die Distanz zu seinen Nachbarn verwenden. Durch die Zertifizierungsstelle wird eine Gleichverteilung der Peers gewährleistet, so dass alle Peers eine in etwa gleich große Distanz zueinander haben sollten.

Behauptet nun ein Peer P, dass er für eine bestimmte Position BP zuständig ist und der Abstand des Peers zu dieser Position überschreitet die maximale Distanz, so wird der Peer erst mal in die Routing Tabelle aufgenommen. Würde der Peer aufgrund der maximalen Distanz nicht aufgenommen, wäre es möglich, dass ein Peer nicht in ein bestimmtes Segment routen kann, weil er für dieses Segment keinen Peer kennt. Allerdings wird in regelmäßigen Abständen weiterhin nach einem Peer gesucht, dessen Abstand die maximale Distanz nicht überschreitet. Zu diesem Zweck kann der Peer auch Anfragen an Positionen senden, die in der Nähe von BP sind und die dafür verantwortlichen Peers nach ihren Nachbarn fragen. Hierdurch erlangt der

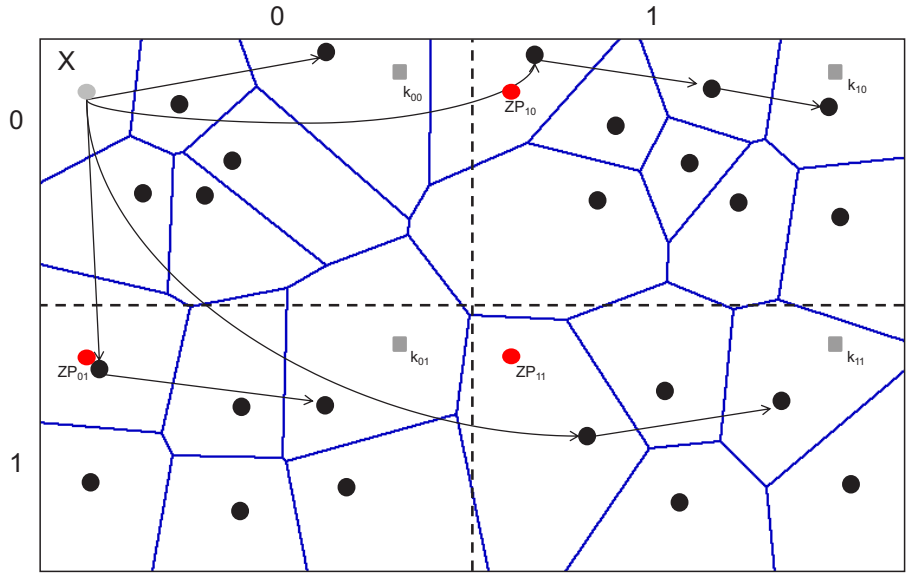


Abbildung 5.6: Symmetrische Replikation mit k=4 und disjunktem Pfadrouting

Peer eine genauere Sicht über die Zuständigkeiten in dem Gebiet. Mit Hilfe dieser Peers kann dann die Zuständigkeit von P verifiziert werden oder der Peer lernt einen Peer kennen, der die maximale Distanz nicht überschreitet.

Wenn ein Peer X alle Replikate k_{xy} eines Objekts anfragen möchte, wird die Anfrage immer direkt zu dem Peer weitergeleitet, der am nächsten an der Zielposition ZP_{xy} ist (siehe Abbildung 5.6). Ein böswilliger Peer bzw. ein Angreifer benötigt daher Peers, die relativ nah an den festgelegten Zielpositionen von X sind, um so die Anfragen für die Replikate über die disjunkten Pfade im ersten Schritt zu verhindern. Da allerdings die Positionen nicht frei gewählt werden können, sondern fest von der Zertifizierungsstelle vergeben werden, ist es sehr unwahrscheinlich, dass ein Angreifer Peers an den notwendigen Positionen hat.

Angesichts der Tatsache, dass Nutzer sich innerhalb der virtuellen Welt bewegen müssen sie, wenn sie sich in eine neue Region bewegen den aktuellen Zustand dieser Region anfragen. Der Zustand der benötigten Region kann dabei auf verschiedenen benachbarten Peers gespeichert sein. Um den Zustand von allen Peers zu erhalten können positionsbezogene Bereichsabfragen, wie in [6] beschrieben, verwendet werden. Hierbei wird jede der k Anfragen für die k Replikate in das jeweilige Zielgebiet gesendet und wird dort an alle relevanten Peers verteilt bevor die Daten gesammelt und zurück gesendet werden.

5.2.4 Bootstrapping und Instandhaltung

Um den in den vorherigen Abschnitten beschriebenen Routing Quader mit entsprechenden Peers zu füllen, müssen der Integrations- und Instandhaltungsmechanismus angepasst werden.

Wenn sich ein Peer neu in das Peer-to-Peer Netzwerk integrieren möchte und der Bootstrapping Service beispielsweise nur einen einzelnen aktiven Peer aus dem Peer-to-Peer Netzwerk zurück gibt, ist es nicht ausgeschlossen, dass der neu beitretende Peer in ein Netzwerk bestehend aus böswilligen Peers integriert wird. Daher muss der Bootstrapping Service mehrere Peers zurückgeben, wobei auch hier angenommen wird, dass in dieser Menge eine ehrliche Mehrheit existiert. Vergleichbar mit Pastry, sendet ein neu beitretender Peer eine Integrationsanfrage für seine eigene Position an die Peers, die er vom Bootstrapping Service erhalten hat. Diese Anfragen werden dann von den Peers an die jeweilige Zielposition weitergeleitet. Allerdings muss bei der Auswahl der Peers darauf geachtet werden, dass die Positionen der ausgewählten Peers nicht so sind, dass sie denselben Peer für das Zielsegment in ihren Routing Quadern haben. Wäre dies der Fall würden alle Anfragen über denselben Peer geleitet, so dass dieser die Integration des Peers beeinflussen kann. Die Antworten auf die Integrationsanfragen beinhalten dann die Voronoinachbarn des neu beitretenden Peers. Anschließend kann der Peer sich zu seinen Voronoinachbarn verbinden und den selbst-stabilisierenden Delaunay-Algorithmus [42] starten, um seine Nachbarschaftsliste instand zu halten.

In Pastry kann ein neu beitretender Peer seine Routing Tabelle mit Einträgen füllen, die er von den Peers erhält, die auf dem Pfad zur Zielposition liegen. Bei VoroStore wäre dies auch möglich, allerdings würden diese Peers bei VoroStore in den meisten Fällen die maximale Distanz überschreiten. Daher wird ein neu beitretender Peer für jede Zielposition in seinem Routing Quader eine Anfrage senden um Peers, die die maximale Distanz nicht überschreiten, zu erhalten. Das bedeutet, dass hierfür ein nennenswerter aber konstanter Zusatzaufwand erzeugt wird, der von der Größe des Routing Quaders abhängig ist.

Aufgrund dessen, dass bei VoroStore Nachrichten rekursiv weitergeleitet werden, können Peers nur in geringem Maße ihren Routing Quader passiv aktualisieren. Der Grund hierfür ist, dass ein Peer beim rekursiven Routing im Vergleich zum iterativen Routing keine neuen Peers kennenlernt, die eventuell zur Aktualisierung verwendet werden könnten. Allerdings sollten die Routing Quader immer die Peers beinhalten, die sich am nächsten an den definierten Zielpositionen befinden. Daher sendet ein neu beigetretener Peer A eine Nachricht an die Peers, die A in ihrem Routing Quader

haben müssten. Sei B der Peer, der am nächsten an einer Zielposition von A ist. In diesem Fall muss A eine Benachrichtigung an B schicken, so dass B den neu beigetretenen Peer A in seinen Routing Quader aufnehmen kann. B kann dann die Nachricht von A an seine Nachbarn weiterleiten, so dass diese überprüfen können, ob A näher an ihren Zielpositionen ist, als die bisher verwendeten Peers. Dementsprechend muss A alle Peers aus seinem Routing Quader über seine Anwesenheit informieren, nachdem er dem Peer-to-Peer Netzwerk beigetreten ist. Das geschieht automatisch wenn A seinen eigenen Routing Quader am Anfang initial füllt. Dieser Benachrichtigungsmechanismus wird ebenfalls verwendet, wenn ein Peer das Peer-to-Peer Netzwerk verlässt, so dass die anderen Peers ihren Routing Quader aktualisieren können.

Wenn ein Peer Anfragen sendet um seinen Routing Quader zu füllen oder zu aktualisieren muss sichergestellt werden, dass sich die Peers für die Zielpositionen im richtigen Segment befinden. Peers, deren Positionen beispielsweise am Rand eines Segments liegen, können auch für Positionen im benachbarten Segment verantwortlich sein. In Abbildung 5.7 ist dieses Problem grafisch dargestellt. Ein Peer (roter Punkt) sendet Anfragen für die Zielpositionen der obersten Schicht (blaue Punkte) seines Routing Quaders. Für die Zielposition im oberen rechten Segment ist allerdings Peer Z verantwortlich, wobei sich Z im oberen linken Segment befindet. In diesem Fall würde der anfragende Peer zwei Peers aus dem oberen linken Segment in seinen Routing Quader aufnehmen. Um das zu verhindern beinhaltet die Antwort auf diese Anfrage auch die Adressen der Nachbarn. Hierdurch wird sichergestellt, dass die im Routing Quader aufgenommen Peers im richtigen Segment liegen auch wenn diese sich nicht am nächsten an der definierten Zielposition befinden. Andernfalls könnten die disjunkten Routingpfade nicht gewährleistet werden.

In dem selbst-stabilisierenden Delaunay Algorithmus, der in VoroStore verwendet wird, tauschen die Peers regelmäßig Informationen über ihre 2-Hop Nachbarschaft mit ihren Nachbarn aus. Daher muss diese 2-Hop Nachbarschaft zusätzlich zur direkten Nachbarschaftliste verwaltet werden. Mit den Informationen aus diesen beiden Listen kann ein Peer seine eigene sowie die Voronoizellen seiner Nachbarn berechnen. Dadurch wird es für böswillige Peers schwieriger Fehlinformationen über die Umgebung zu verbreiten. Wenn ein einziger böswilliger Peer beispielsweise über seine Nachbarschaft lügt, können seine Nachbarn dies erkennen. Um erfolgreich die Netzwerkstruktur in einer Region zu fälschen, müsste eine Gruppe von böswilligen Peers in einer Region kooperieren. Da allerdings Peers ihre Positionen nicht beeinflussen können, ist die Wahrscheinlichkeit, dass mehr als die Hälfte der Voronoinachbarn eines Peers Betrüger sind sehr gering.

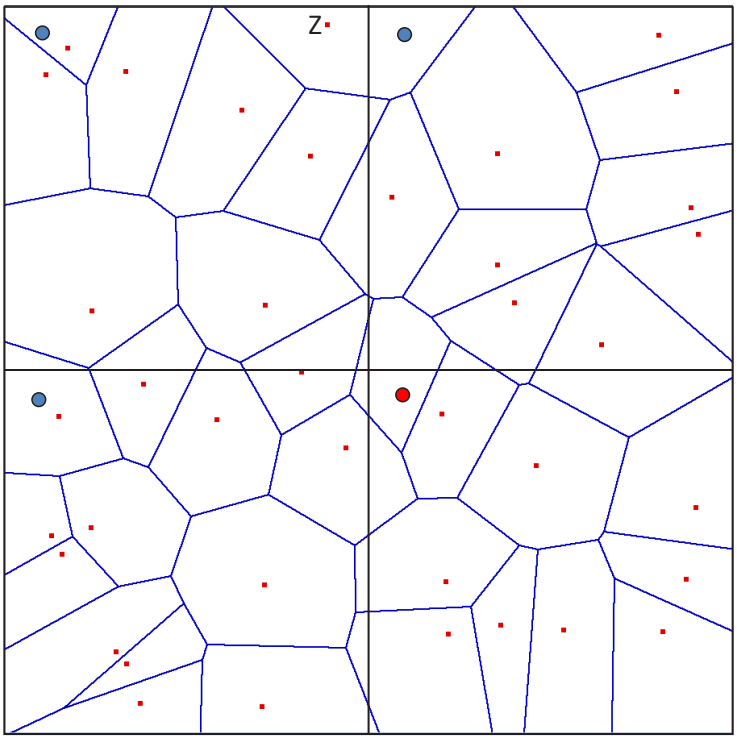


Abbildung 5.7: Zuständigkeit an den Segmentgrenzen

5.3 Verwandte Arbeiten

In diesem Abschnitt wird eine Reihe von verwandten Ansätzen und Arbeiten beschrieben und mit VoroStore verglichen.

Past [72] und OceanStore [49] realisieren beide einen verteilten Datenspeicher. Ähnlich wie VoroStore verwenden sie Redundanz, um die Verfügbarkeit der Daten zu gewährleisten. Beide Ansätze basieren auf verteilten Hashtabellen. Hierbei wird durch eine Hashfunktion bestimmt wo ein Objekt gespeichert wird. Durch die Verwendung einer Hashfunktion geht der Zusammenhang von Objekten, die in der virtuellen Welt nah bei einander sind, verloren. Daher unterstützen Ansätze, die auf verteilten Hashtabellen basieren keine effizienten positionsbezogenen Bereichsabfragen, wie es in einem Positions-basierten Netzwerk möglich ist.

Geostry [47] ist ein Framework für Kontext-basierte Informationen, das auf dem Peer-to-Peer Netzwerk Pastry basiert. Hierbei ermöglicht es Geostry Daten aus einem zweidimensionalen Positionsraum in einem eindimensionalen Netzwerk zu speichern. Zu diesem Zweck verwendet Geostry raumfüllende Kurven, die einen zweidimensionalen Positionsraum auf einen eindimensionalen Adressraum projizieren. Durch diese Projektion ist es allerdings möglich, dass Daten, die im zweidimensio-

nalen Raum nah beieinander sind, im eindimensionalen Raum weiter voneinander entfernt sind. Hierdurch können positionsbezogene Bereichsabfragen nicht so effizient wie mit einem Positions-basierten Netzwerk umgesetzt werden.

Das Peer-to-Peer System CAN [64] hingegen erlaubt es Daten mit Positionsbezug zu speichern. CAN unterstützt mehrdimensionale Adressräume, die durch Rechtecke zerlegt werden. Jeder Peer ist für eines dieser Rechtecke verantwortlich. Allerdings existieren in CAN keine Mechanismen, um den Einfluss von böswilligen Peers zu reduzieren und zusätzlich bietet CAN kein logarithmisches Routing.

Im Bereich Peer-to-Peer-basierter virtueller Welten existiert eine Reihe von Arbeiten, die auf Voronoi-Zerlegungen und Delaunaygraphen basieren. Das von Hu et al. vorgestellte Peer-to-Peer Netzwerk VON [41] verwendet eine Voronoi-Zerlegung, um einen Delaunaygraphen zu bilden. VoroNet [10] ist ebenso ein Peer-to-Peer Netzwerk, dessen Struktur auf einem Delaunaygraphen basiert und weit entfernte Nachbarn verwendet, um die Anzahl der Routing Schritte zu reduzieren. Bei Nomad [66] und [16, 28] handelt es um ähnliche Ansätze. Die Voronoi-Zerlegung wird hierbei verwendet, um die Objekte der virtuellen Welt den teilnehmenden Peers zuzuordnen [15, 40, 65]. Durch die Verwendung von positionsbezogenen Bereichsabfragen [6, 55] kann der aktuelle Zustand bestimmter Objekte in einer Region effizient ermittelt werden. Im Gegensatz zu VoroStore verwenden diese Ansätze die dynamische Position eines Nutzer innerhalb der virtuellen Welt für die Position innerhalb des Peer-to-Peer Netzwerks. Hierdurch entstehen hohe Instandhaltungskosten für das Peer-to-Peer Netzwerk, da die Struktur angepasst werden muss, sobald sich ein Nutzer innerhalb der virtuellen Welt bewegt. Die Nutzer in diesen Ansätzen haben zusätzlich die Möglichkeit zu bestimmen für welche Objekte sie verantwortlich sind, da sie ihre Position bestimmen können. Des Weiteren verwendet keiner dieser Ansätze einen selbst-stabilisierenden Algorithmus, um die Netzwerkstruktur aufrecht zu erhalten.

Keiner der betrachteten Ansätze beinhaltet Verfahren um die Verfügbarkeit der Daten in Verbindung mit böswilligen Peers sicherzustellen. VoroStore hingegen erweitert den Ansatz symmetrischer Replikation mit disjunktem Pfadrouting [77] für Voronoi-basierte Peer-to-Peer Netzwerke. Um betrügerische Updates für die Routing Quader zu verhindern, verwendet VoroStore Zielpositionen für jeden Eintrag [18]. Durch die Verwendung einer zentralen Zertifizierungsstelle werden *Sybil Angriffe* [24] erschwert und durch die feste Zuordnung der Positionen haben böswillige Peers keinen Einfluss auf ihre Zuständigkeit.

5.4 Evaluation

Nachdem in den vorherigen Abschnitten VoroStore beschrieben wurde, werden im folgenden Abschnitt die Ergebnisse der Evaluation dargestellt. Zu diesem Zweck wurde VoroStore prototypisch unter Verwendung der in Kapitel 4 vorgestellten peers@play Architektur und der Programmiersprache Microsoft .NET C# 4.0 implementiert. Die Evaluation umfasst hierbei verschiedene Aspekte, wie die Größe der Voronoizellen, die Anzahl der Voronoinachbarn sowie die Wahrscheinlichkeit einer erfolgreichen Datenabfrage bei symmetrischer Replikation und disjunktem Pfadrouting. Im Folgenden werden die einzelnen Ergebnisse ausführlich betrachtet.

Größe der Voronoizellen

Der erste Aspekt, der im Rahmen der Evaluation betrachtet wird, betrifft die Größe der Voronoizellen. Je größer die Voronoizelle eines Peers ist, desto häufiger kommt dieser Peer in den Routing Quadern anderer Peers vor. Das führt dazu, dass mehr Nachrichten über diesen Peer geroutet werden und dieser Peer zum einen mehr Einfluss gewinnt und zum anderen stärker belastet wird. Um dies zu verhindern, sollten die Voronoizellen der Peers in etwa die gleiche Größe haben.

Um die Größe der Voronoizellen zu bestimmen wurde VoroStore in unterschiedlichen Konfigurationen (Anzahl an Peers) ausgeführt und die Größe der Voronoizellen ermittelt. Hierbei wurde ein Positionsraum der Größe $256 \cdot 256$ (b=2, l=8) verwendet und die Positionen der Peers wurden zufällig gewählt. Jede Konfiguration wurde 10-mal durchgeführt und die Ergebnisse gemittelt. In Tabelle 5.1 sind die gemittelten Ergebnisse dargestellt. Die Zeilen $Q1$ und $Q3$ geben hierbei die Werte für das untere und obere Quartil an, welche 50% der ermittelten Werte umfassen.

Die Standardabweichung der Ergebnisse zwischen den einzelnen Testdurchläufen ist in Tabelle 5.2 zusammengefasst.

Die Ergebnisse zeigen, dass bei einer geringen Peerdichte die Größe der Voronoizellen stärker variiert und dass mit steigender Anzahl an Peers, die Abweichungen bei den Zellengrößen geringer werden. Das gleiche gilt für die Standardabweichung zwischen den einzelnen Testläufen.

Die Schwankungen in der Größe der Voronoizellen resultieren aus der zufälligen Vergabe der Peer Positionen. Wenn man beispielsweise die Positionen in Form eines Gitters vergibt, so haben alle Voronoizellen fast dieselbe Größe. Daher muss bei der Vergabe der Positionen versucht werden, diese möglichst gleichmäßig zu vergeben.

	500	1000	5000	10000
Dichte (%)	0,76	1,53	7,63	15,26
Größe (berechnet)	131,072	65,536	13,1072	6,5536
Minimum	12,2	4	1	1
Maximum	408,1	221,9	54,9	27,7
Q1	79,1	40,1	8	4
Q2 (Median)	119,1	59,8	12	6
Q3	170,1	84,9	17	8
Standardabweichung	69,6	34,6	6,8	3,3

Tabelle 5.1: Größe der Voronoizellen

	500	1000	5000	10000
Minimum	4,4	1,7	0	0
Maximum	46,9	19,0	6,9	2,3
Q1	3,3	0,9	0	0
Q2 (Median)	2,4	1,0	0	0
Q3	3,1	1,5	0	0
Standardabweichung	4,2	0,6	0,1	0

Tabelle 5.2: Standardabweichung der einzelnen Testdurchläufe - Voronoizellen

Da die Positionen über die Zertifizierungsstelle vergeben werden, kann an dieser Stelle Einfluss auf die Verteilung genommen werden.

Anzahl der Voronoinachbarn

Die Anzahl der Nachbarn in einem Voronoi-basierten Peer-to-Peer Netzwerk ist nicht begrenzt. Je mehr Nachbarn ein Peer hat, desto höher ist der Verwaltungsaufwand für diesen Peer, da zu jedem Nachbarn eine Verbindung aufrecht gehalten werden muss. Um die Anzahl der Nachbarn zu bestimmen wurde VoroStore in unterschiedlichen Konfigurationen ausgeführt. Wie im vorherigen Abschnitt wurde ein Positionsraum der Größe $256 \cdot 256$ (b=2, l=8) verwendet und die Positionen der Peers wurden zufällig gewählt.

In diesem Szenario wurde jede Konfiguration 5-mal durchführt und die Ergebnisse und die Standardabweichung zwischen den einzelnen Testläufen sind in Tabelle 5.3 dargestellt. Die dargestellten Ergebnisse zeigen, dass im Gegensatz zu der Größe der

	100	300	500	100_{ST}	300_{ST}	500_{ST}
Durchschnitt	5,71	5,88	5,92	0,04	0,01	0,01
Minimum	3,2	3	3	0,45	0	0
Maximum	9	10,4	10,6	0,71	0,55	0,89
Q1	5	5	5	0	0	0
Q2 (Median)	5,6	6	6	0,55	0	0
Q3	6,6	6,8	7	0,55	0,45	0
Standardabweichung	1,29	1,34	1,34	0,10	0,09	0,05

Tabelle 5.3: Anzahl der Voronoinachbarn

Voronoizelle die Peerdichte keinen signifikanten Einfluss auf die Anzahl der Nachbarn hat. Ebenso zeigen die Ergebnisse der einzelnen Testdurchläufe keine großen Schwankungen. Betrachtet man den Durchschnitt sowie den Median so hat jeder Peer bei VoroStore 6 Nachbarn.

Die Menge der Voronoinachbarn bei VoroStore ist konzeptionell vergleichbar mit dem Leaf-Set, das von Pastry verwendet wird. Die Größe des Leaf-Sets bei Pastry ist allerdings konfigurierbar. Typische Werte für die Größe sind 4 oder 8. Betrachtet man die Anzahl der Voronoinachbarn und die Größe des Leaf-Sets so ist der Verwaltungsaufwand eines Peers in beiden Ansätzen vergleichbar.

Zielpositionen

VoroStore verwendet, wie in Abschnitt 5.2.3 beschrieben, Zielpositionen für die einzelnen Zellen des Routing Quaders. Da nicht für jede Zielposition ein Peer mit genau der geforderten Position existieren muss, werden auch Peers verwendet, die am nächsten an der geforderten Position liegen. Damit böswillige Peers nicht einfach dafür sorgen können, dass sie in die Routing Quadern von ehrlichen Peers aufgenommen werden, wird eine maximale Distanz verwendet um zu entscheiden, ob ein Peer für eine Position verantwortlich sein kann. VoroStore verwendet hierbei eine dynamische Distanz, die von jedem Peer selbst berechnet wird. Hierfür ermittelt jeder Peer die Abstände zu seinen Voronoinachbarn und bildet den Durchschnitt über die Abstände.

Um zu überprüfen, ob der durchschnittliche Abstand zu den Nachbarn repräsentativ für die Abstände im gesamten Netzwerk ist, wurden die Abstände in verschiedenen Konfigurationen ermittelt. Es wurden hierfür zwei verschiedene Werte bestimmt. Der erste Wert ist der Abstand zwischen allen Voronoinachbarn im Peer-to-Peer

Netzwerk (globaler Abstand). Der zweite Wert ist der durchschnittliche Abstand eines Peers zu seinen Nachbarn (lokaler Abstand).

	100	300	500	100_{ST}	300_{ST}	500_{ST}
Durchschnitt	31,42	17,65	13,57	0,83	0,16	0,17
Minimum	10,31	4,53	3,25	2,15	1,13	0,45
Maximum	85,74	68,15	58,19	15,18	8,36	11,62
Q1	22,77	12,83	10,03	1,57	0,27	0,19
Q2 (Median)	28,80	16,32	12,50	1,34	0,24	0,10
Q3	36,42	20,33	15,46	1,35	0,31	0,14
Standardabweichung	13,29	7,97	6,41	1,45	0,31	1,03

Tabelle 5.4: Abstand zwischen den Voronoinachbarn (lokal)

Um beide Werte zu bestimmen wurde VoroStore mit 100, 300 und 500 Peers ausgeführt. Jede Messung wurde 5-mal wiederholt und die Abweichung zwischen den einzelnen Tests bestimmt. Die Ergebnisse sind in Tabelle 5.4 und 5.5 zusammengefasst und zeigen, dass der globale und lokale Abstand kaum voneinander abweichen. Das bedeutet, dass der Abstand zu den eigenen Nachbarn ein guter Richtwert ist um zu entscheiden, ob ein Peer für eine Position zuständig sein kann.

	100	300	500	100_{ST}	300_{ST}	500_{ST}
Durchschnitt	32,59	18,49	11,54	0,67	0,21	1,73

Tabelle 5.5: Abstand zwischen den Voronoinachbarn (global)

Allerdings gibt es auch Peers, deren lokaler Abstand deutlich über oder unter dem globalen Abstand liegt (siehe Tabelle 5.4 Minimum und Maximum). In den Fällen, in denen der lokale Abstand größer ist als der Globale, bedeutet das, dass der Peer eventuell einen Peer in seine Tabelle aufnimmt, der gemessen am globalen Abstand zu weit entfernt ist. Das bedeutet, dass es eventuell einen Peer geben könnte, der noch näher an der definierten Zielposition ist. In den Fällen, in denen der lokale Abstand geringer ist als der Globale, hat das zur Folge, dass der Peer häufiger versucht neue Peers zu finden, die einen geringen Abstand zur definierten Zielposition haben. Dies hat zur Folge, dass der Aufwand zur Aktualisierung des Routing Quaders unnötigerweise steigt.

Größe des Routing Quaders

Nachdem im letzten Abschnitt die Zielpositionen näher brachtet wurden, wird im Folgenden der Routing Quader näher betrachtet. Die allgemeine Größe des Routing Quaders wird mit $b^2 \cdot l$ berechnet. Allerdings sind die Zellen, für die der Peer selbst zuständig ist nicht besetzt, so dass sich maximal $(b^2-1)\cdot l$ Peers im Routing Quader befinden (siehe Abbildung 5.4). Betrachtet man die maximale Anzahl an Einträgen bei einer verwendeten Zahlenbasis von 2 und einer Länge von 8 Ziffern, ergeben sich maximal 24 Einträge im Routing Quader. Allerdings existiert nicht für jede Zielposition ein Peer, der die Bedingung für den Präfix erfüllt, so dass einige Einträge leer sein können.

Um die Anzahl der Peers in den Routing Quadern zu bestimmen, wurde VoroStore mit verschiedenen Peerdichten ausgeführt und bei jedem Peer die Anzahl der Peers in dessen Routing Quader ermittelt. Die Ergebnisse sind in Tabelle 5.6 dargestellt.

	100	300	500	100_{ST}	300_{ST}	500_{ST}
Durchschnitt	12,04	14,43	15,46	0,14	0,05	0,12
Minimum	7,8	8	8	1,30	1,22	1,41
Maximum	16,8	19,8	21	0,84	1,30	0
Q1	10,8	13	14	0,44	0	0
Q2 (Median)	12	14	15,2	0	0	0,45
Q3	13,2	15,4	16,6	0,45	0,55	0,55
Standardabweichung	1,89	1,86	1,81	0,22	0,14	0,07

Tabelle 5.6: Größe des Routing Quaders

Hierbei wird deutlich, dass mit steigender Peerdichte auch die Anzahl der Peers in den Routing Quadern steigt. Je mehr Peers im Netzwerk sind, desto höher ist die Wahrscheinlichkeit, dass ein Peer existiert, der einen geeigneten Präfix hat. Bei 100 Peers, was einer Peerdichte von 0,76% entspricht, ist der Routing Quader im Schnitt bereits zu 50% mit Peers besetzt.

Disjunktes Routing und Replikation

Im folgenden Abschnitt wird evaluiert, welchen Einfluss böswillige Peers auf den Erfolg einer Anfrage haben. Zu diesem Zweck wird zuerst eine theoretische Abschätzung betrachtet.

Hierbei wird davon ausgegangen, dass sich P Peers im Peer-to-Peer Netzwerk befinden. Die prozentuale Anzahl böswilliger Peers wird mit x bezeichnet, so dass insgesamt $X = P \cdot x$ böswillige Peers im Netzwerk sind. Der Positionsraum wird durch die verwendete Zahlenbasis b und die Länge der Positionen l bestimmt. Die Länge des Routingpfades rp wird mit $log_{2b}(P)$ berechnet.

Um den Einfluss von böswilligen Peers zu berechnen, muss zuerst berechnet werden, wie hoch die Wahrscheinlichkeit ist, dass auf einem Routingpfad kein böswilliger Peer liegt. Zu diesem Zweck wird die hypergeometrische Verteilung verwendet. Hierbei wird ermittelt, wie viele Objekte beim Ziehen aus einer Menge ohne Zurücklegen eine bestimmte Eigenschaft aufweisen. Die Wahrscheinlichkeit, dass kein böswilliger Peer auf dem Routingpfad (EP) liegt wird durch die folgende Formel berechnet:

$$EP = \frac{\binom{X}{0} \cdot \binom{P-X}{log_{2b}(P)}}{\binom{P}{log_{2b}(P)}}$$

Nachdem die Wahrscheinlichkeit dafür, dass kein böswilliger Peer auf einem Routingpfad liegt, berechnet wurde, wird jetzt berechnet wie hoch die Wahrscheinlichkeit ist, dass auf mehr als der Hälfte der Routingpfade zu den Replikaten kein böswilliger Peer liegt. Wenn k Replikate im Peer-to-Peer Netzwerk existieren, dann muss mindestens die Hälfte aller Pfade zu den Replikaten vollständig aus ehrlichen Peers bestehen. Das bedeutet, dass nur auf höchstens z Pfaden, mit $z < \frac{k}{2}$ ein oder mehr böswillige Peer liegen dürfen. Die Wahrscheinlichkeit hierfür wird mit Hilfe der kumulierten hypergeometrischen Verteilung berechnet:

$$\sum_{i=0}^{z} \frac{\binom{RP \cdot (1-EP)}{k} \cdot \binom{RP \cdot EP}{k-i}}{\binom{RP}{k}}$$

In Tabelle 5.7 sind die berechneten Erfolgswahrscheinlichkeiten für verschieden große Peer-to-Peer Netzwerke und verschiedene Replikationsfaktoren dargestellt.

Die Ergebnisse zeigen, dass mit steigender Anzahl an Peers die Erfolgswahrscheinlichkeit sinkt. Das liegt daran, dass mit steigender Anzahl an Peers auch die Länge des Routingpfades steigt. Je länger der Routingpfad wird, desto größer ist die Wahrscheinlichkeit, dass ein böswilliger Peer auf diesem Pfad liegt. Betrachtet man die Erfolgswahrscheinlichkeit bei einem Replikationsfaktor von 9 und einem Anteil von böswilligen Peers von 5% liegt die Erfolgswahrscheinlichkeit bei über 97%.

Böswillige Peers (%)	Gesamtanzahl Peers 100	500	2000
1	99,29	98,87	98,36
2	96,97	95,46	93,67
3	93,45	90,42	87,09
4	89,05	84,47	79,47
5	84,05	77,91	71,42
10	56,55	45,45	35,71
20	17,84	10,03	5,39

b=2, l=8, k=4

Böswillige Peers (%)	Gesamtanzahl Peers 100	500	2000
1	100	100	100
2	100	99,99	99,97
3	99,98	99,92	99,75
4	99,91	99,64	99,00
5	99,72	98,97	97,37
10	93,96	84,77	72,08
20	52,08	27,59	12,49

b=3, l=8, k=9

Tabelle 5.7: Wahrscheinlichkeiten erfolgreicher Anfragen in Verbindung mit böswilligen Peers

Zusätzlich zu der theoretischen Betrachtung der Erfolgswahrscheinlichkeit wurde diese mit VoroStore ermittelt. Hierbei wurden Peer-to-Peer Netzwerke mit 100 und 500 Peers (N) bei verschiedenen Replikationsfaktoren und unterschiedlichen Anteilen an böswilligen Peers ausgewertet. Zu diesem Zweck wurden 500 Daten an verschiedenen Positionen im Netzwerk gespeichert und anschließend wurden die Daten von 500 verschiedenen Peers aus angefragt. Jede Testreihe wurde 5-mal durchgeführt und die Ergebnisse sind in den Tabellen 5.8, 5.9, 5.10 und 5.11 zusammengefasst. Die Tabellen zeigen für unterschiedlich große Netzwerke und unterschiedlichen Anteilen an böswilligen Peers, wie viele der 500 Anfragen erfolgreich waren. In den letzten beiden Zeilen einer jeden Tabelle, befinden sich der mit VoroStore ermittelte durchschnittliche Wert für eine erfolgreiche Anfrage sowie der Wert aus der theoretischen Betrachtung.

Vergleicht man die Ergebnisse der theoretischen Berechnung mit den ermittelten Ergebnissen von VoroStore, so liegen die Werte der ermittelten Ergebnisse über denen der theoretischen Berechnung. Die Gründe hierfür sind zum einen, dass die in der Berechnung verwendete Länge des Routingpfades eine *worst case* Abschätzung ist und zum anderen dass die Einschränkungen des Routing Quaders nicht berücksichtigt sind. Hierdurch ergibt sich bei der theoretischen Berechnung eine deutlich pessimistischere Abschätzung.

	Anteil böswilliger Peers		
Erfolgreiche Anfragen	1%	5%	10%
Minimum	500	486	464
Maximum	500	500	493
Median	500	497	483
Standardabweichung	0	6,52	12,82
Durchschnitt	500	494	480
Durchschnitt (%)	100	98,8	96
Theoretisch (%)	99,29	84,05	56,55

Tabelle 5.8: Erfolgreiche Anfragen bei unterschiedlichen Anteilen böswilliger Peers (N=100, b=2, l=8, k=4)

	Anteil böswilliger Peers		
Erfolgreiche Anfragen	1%	5%	10%
Minimum	499	488	473
Maximum	500	497	479
Median	500	493	477
Standardabweichung	0,45	2,75	2,59
Durchschnitt	499	492	476
Durchschnitt (%)	99,8	98,6	95,2
Theoretisch (%)	98,87	77,91	45,45

Tabelle 5.9: Erfolgreiche Anfragen bei unterschiedlichen Anteilen böswilliger Peers (N=500, b=2, l=8, k=4)

	Anteil böswilliger Peers		
Erfolgreiche Anfragen	1%	5%	10%
Minimum	500	500	500
Maximum	500	500	500
Median	500	500	500
Standardabweichung	0	0	0
Durchschnitt	500	500	500
Durchschnitt (%)	100	100	100
Theoretisch (%)	100	99,72	93,96

Tabelle 5.10: Erfolgreiche Anfragen bei unterschiedlichen Anteilen böswilliger Peers (N=100, b=3, l=8, k=9)

	Anteil böswilliger Peers		
Erfolgreiche Anfragen	1%	5%	10%
Minimum	500	500	499
Maximum	500	500	500
Median	500	500	500
Standardabweichung	0	0	0,55
Durchschnitt	500	500	499,6
Durchschnitt (%)	100	100	99,92
Theoretisch (%)	100	98,97	84,77

Tabelle 5.11: Erfolgreiche Anfragen bei unterschiedlichen Anteilen böswilliger Peers (N=500, b=3, l=8, k=9)

Kosten

VoroStore erlaubt es Daten verlässlich mit Positionsbezug zu speichern. Die Verlässlichkeit der Daten kann hierbei nur mit einer bestimmten Wahrscheinlichkeit gewährleistet werden kann. Die Wahrscheinlichkeit wird über den gewählten Replikationsgrad k beeinflusst.

Durch die Replikation, die Instandhaltung des Routing Quaders und die k-fache Anfrage für die Replikate wird der Nachrichtenaufwand erhöht. Wenn ein Peer das Peer-to-Peer Netzwerk betritt oder verlässt muss er an jede Zielposition seines Routing Quaders eine Nachricht schicken. Zusätzlich muss anstatt einer Nachricht für eine Anfrage k Anfragen gesendet werden.

Allerdings handelt es sich bei den Replikationsanfragen und den Benachrichtigungen um einen konstanten Aufwand. Der Aufwand für einen Peer hängt somit von diesen Konstanten sowie von der Verteilung der Peers ab, aber nicht von der Gesamtanzahl der Peers.

5.5 Zusammenfassung

In diesem Kapitel wurde VoroStore vorgestellt. VoroStore erlaubt es Daten mit Positionsbezug an verschiedenen Positionen in einem Peer-to-Peer Netzwerk zu speichern und abzufragen. Durch die Kombination von einer k-fachen symmetrischen Replikation und dem disjunktem Pfadrouting, kann die Verfügbarkeit und die Integrität

der gespeicherten Daten mit einer bestimmten Wahrscheinlichkeit gewährleistet werden. Zu diesem Zweck wurde das Präfix-basierte Routing Verfahren von Pastry für einen zweidimensionalen Positionsraum erweitert. Der hierdurch entstandene Routing Quader ermöglicht in einem zweidimensionalen Positionsraum ein Routing Verfahren mit einer logarithmischen Anzahl an Routing Schritten. Um den Einfluss von böswilligen Peers zu beschränken, wurden die einzelnen Zellen des Routing Quaders mit speziellen Zielpositionen versehen. Durch diese Zielpositionen und der festen Positionsvergabe durch die Zertifizierungsstelle, ist es böswilligen Peers nicht möglich ihr Vorkommen in den Routing Quadern von ehrlichen Peers zu erhöhen. In der Evaluation wurde gezeigt, dass bei VoroStore die Wahrscheinlichkeit für eine erfolgreiche Anfrage bei über 97% liegt, wenn 5% der teilnehmenden Peers böswillig sind.

Um den Einfluss von böswilligen Peers zu reduzieren, erzeugt VoroStore einen nicht zu vernachlässigen zusätzlichen Aufwand im Vergleich zu nicht redundanten Speicherlösungen. Allerdings ist der Aufwand nicht abhängig von der Gesamtzahl der Peers, sondern vom gewählten k sowie der Größe des Positionsraums, da dieser die Größe des Routing Quaders bestimmt. Bei beiden Größen handelt es sich um konstanten Aufwand, der nicht von der Anzahl der Peers abhängt.

Kapitel 6

Zusammenfassung und weiterer Forschungsbedarf

Peer-to-Peer-basierte Anwendungen reduzieren im Vergleich zu Anwendungen, die auf einer Client-Server Architektur basieren, die Notwendigkeit von dedizierten Servern. Allerdings ergibt sich bei der Entwicklung von Peer-to-Peer-basierten Anwendungen eine Reihe von Herausforderungen. Diese Herausforderungen können von Anwendung zu Anwendung variieren, wobei einige Herausforderungen von allen Peer-to-Peer-basierten Anwendungen gelöst werden müssen. Die vorliegende Arbeit hat hierbei verschiedene Herausforderungen aus unterschiedlichen Themengebieten näher betrachtet.

Im ersten Teil dieser Arbeit wurde die allgemeine Problematik des Verbindungsaufbaus der Peers in Verbindung mit NAT Routern ausführlich betrachtet. Hierbei wurde ein neues TCP Hole Punching Verfahren namens SYNI vorgestellt. Das Verfahren initiiert eine TCP Verbindung durch die Injektion eines TCP SYN Pakets. Damit das Verfahren erfolgreich eine TCP Verbindung herstellen kann, müssen die involvierten NAT Router bestimmte Anforderungen erfüllen. Jedes TCP Hole Punching Verfahren hat hierbei andere Anforderungen an die involvierten NAT Router, so dass für jeden Verbindungsaufbau entschieden werden muss, welches Verfahren eingesetzt werden sollte. Zu diesem Zweck wurde das in dieser Arbeit vorgestellte MFB Protokoll entwickelt. Dieses Protokoll ermöglicht es die relevanten Eigenschaften eines NAT Routers zu bestimmen und anhand eines Regelwerks zu entscheiden, welches Verfahren verwendet werden sollte. Für die Evaluation des TCP Hole Punching Verfahrens SYNI und des MFB Protokolls wurde ein spezielles Testsystem entwickelt, so dass beide Ansätze mit verschiedenen NAT Routern evaluiert werden konnten. Die Evaluation zeigt, dass mit dem Verfahren SYNI in $98,7\%$ der Fälle, in denen Hole Punching Verfahren eingesetzt werden konnten, erfolgreich eine TCP

Verbindung hergestellt werden konnte. Der zweite Teil der Evaluation zeigt, dass unter Verwendung der durch das MFB Protokoll ermittelten Eigenschaften und des Regelwerks in allen Fällen im ersten Versuch erfolgreich eine Verbindung hergestellt werden konnte.

Der zweite Themenschwerpunkt dieser Arbeit betrachtet die peers@play Architektur, eine Softwarearchitektur für komplexe Peer-to-Peer Systeme. Hierbei wird eine Peer-to-Peer Anwendung in verschiedene Schichten und Komponenten aufgeteilt, um die Wiederverwendbarkeit und Austauschbarkeit von bereits entwickelten Ansätzen zu ermöglichen. Zusätzlich unterstützt die peers@play Architektur mehrere zeitgleich aktive Peer-to-Peer Netzwerke. Hierdurch ist es möglich unterschiedliche Peer-to-Peer Netzwerke für verschiedene Zwecke einzusetzen. Im peers@play Projekt werden beispielsweise unterschiedliche Netzwerke für die Datenspeicherung und die dynamischen Positionen der Teilnehmer verwendet. Eine weitere Eigenschaft der peers@play Architektur ist, dass derselbe Quelltext für Simulationen als auch für die Produktivanwendung verwendet werden kann. Dies wird durch die Verwendung des Programmiermodells Gears4Net und eines speziellen Schedulers, Zeitgebers und einer Kommunikationskomponente ermöglicht. Durch die Aufteilung in Schichten und Komponenten und die Möglichkeit der Simulation ist es ohne großen Aufwand möglich neue Ansätze zu integrieren und das neue Gesamtsystem zu evaluieren. Die Evaluation der peers@play Architektur zeigt, dass diese nur eine geringe zusätzliche Verarbeitungszeit verursacht. Ebenso hat die peers@play Architektur einen geringen Speicherverbrauch, so dass eine Vielzahl an Peers auf einem einzelnen Rechner simuliert werden kann.

Das dritte und letzte Themengebiet, das in dieser Arbeit betrachtet wird, umfasst die verlässliche Datenverteilung in Peer-to-Peer Netzwerken. Das hierbei vorgestellte Konzept VoroStore verwendet Voronoi-Diagramme, um den verwendeten Positionsraum aufzuteilen sowie ein spezielles Routing- und Replikationsverfahren. Das Replikationsverfahren sorgt dafür, dass die Daten an verschiedenen Positionen im Peer-to-Peer Netzwerk gespeichert werden. Die Positionen werden hierbei so gewählt, dass diese mit dem Routingverfahren auf disjunkten Routingpfaden erreicht werden können. Hierfür verwendet VoroStore einen Routing Quader, der auf dem Prinzip der Routing Tabelle von Pastry basiert. Damit böswillige Peers ihr Vorkommen in den Routing Quadern von ehrlichen Peers nicht erhöhen können, verwendet der Routing Quader spezielle Zielpositionen für jeden Eintrag. Um den Einfluss von böswilligen Peers zu bestimmen wurde dieser zum einen theoretisch abgeschätzt und zum anderen mit einer prototypischen Implementation gemessen. Hierbei wurde gezeigt, dass je nach Größe des Netzwerks und einem Anteil böswilliger Peers von 5% und einem

Replikationsgrad von 4 die Wahrscheinlichkeit für eine erfolgreiche Anfrage in der Theorie und Praxis bei über 97% liegt.

Weiterer Forschungsbedarf Die vorliegende Arbeit betrachtet eine Reihe von Fragestellungen und Herausforderungen bei der Entwicklung von Peer-to-Peer-basierten Anwendungen. Allerdings bleiben einzelne Detailfragen offen beziehungsweise führen die in dieser Arbeit vorgestellten Ansätze weiter.

Die Ergebnisse der im ersten Kapitel vorgestellten Verfahren basieren auf den für die Evaluation verwendeten NAT Routern. In diesem Bereich könnten durch eine größere Anzahl an NAT Routern neue Erkenntnisse über weitere Verhaltensweisen ermittelt werden. Ebenso könnte überprüft werden, ob die Verhaltensweisen, die in dieser Arbeit ermittelt wurden, von einer größeren Menge an NAT Routern gezeigt werden. In diesem Fall könnte es interessant sein neue NAT Traversal Verfahren basierend auf diesen Verhaltensweisen zu entwickeln. In [87] wurden zwei NAT Traversal Verfahren vorgestellt, die die Application Level Gateway (ALG) Funktion einiger NAT Router verwenden um eine TCP Verbindung aufzubauen. Sollten diese Verfahren mit einer Vielzahl an NAT Routern funktionieren, so könnten diese Verfahren in das in dieser Arbeit vorgestellte Regelwerk integriert werden.

Ein Punkt, der im Rahmen der peers@play Architektur näher betrachtet werden kann, ist ein dynamischer Wechsel der in den Schichten verwendeten Ansätze. Dieser Wechsel ist möglich, da die Schichten nur über einen asynchronen Nachrichtenaustausch miteinander kommunizieren. Um einen Ansatz auszutauschen müssen nur die Nachrichtenwarteschlangen angehalten werden und sobald der Ansatz ausgetauscht ist wieder gestartet werden. Hierdurch wäre es möglich das Peer-to-Peer Netzwerk abhängig von der Anzahl der Teilnehmer zu wechseln.

Als weiterführende Arbeit im Rahmen von VoroStore könnte man höher dimensionale Voronoi-Zerlegung verwenden, um so zusätzliche Dimensionen für Raum, Zeit und Benutzer zu erhalten. Dadurch könnte man Daten noch spezifischer Anfragen, was allerdings auch den Aufwand erhöht. Des Weiteren könnte man VoroStore um Vertraulichkeit und Zugriffskontrolle erweitern. Informationen, die ein Nutzer hat, sollten sich nur auf die Region beziehen, in der der Nutzer sich auch gerade befindet. Wenn ein Nutzer über mehr Informationen über die virtuelle Welt verfügt als andere, kann er sich damit eventuell einen Vorteil gegenüber anderen Nutzer verschaffen. Daher könnte man den Zugriff auf die gespeicherten Daten so einschränken, dass nur Nutzer, die in einer bestimmten Region sind, die Daten aus dieser Region abfragen und modifizieren können.

Literaturverzeichnis

[1] Published on the WWW at `http://www.cs.unm.edu/~riesen/SysComps/Questions.html`. (retrieved on 2012-03-26).

[2] CrypTool 2.0. Published on the WWW at `http://www.cryptool2.vs.uni-due.de`. (retrieved on 2011-07-01).

[3] Peers@Play. Published on the WWW at `http://www.peers-at-play.org`. (retrieved on 2011-07-01).

[4] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/. (retrieved on 2011-07-01).

[5] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth. The Essence of P2P: A Reference Architecture for Overlay Networks. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 11–20, Washington, DC, USA, 2005. IEEE Computer Society.

[6] M. Albano, L. Ricci, M. Baldanzi, and R. Baraglia. VoRaQue: Range queries on Voronoi overlays. In *Computers and Communications, 2008. ISCC 2008. IEEE Symposium on*, pages 495 –500, July 2008.

[7] F. Audet and C. Jennings. Network Address Translation (NAT) Behavioral Requirements for Unicast UDP, Jan. 2007.

[8] F. Aurenhammer. Voronoi Diagrams – A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, 23(3):345–405, 1991. Habilitationsschrift. [Report B 90-09, FU Berlin, Germany, 1990].

[9] I. Baumgart, B. Heep, and S. Krause. OverSim: A scalable and flexible overlay framework for simulation and real network applications. In *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, pages 87 –88, Sept. 2009.

[10] O. Beaumont, A.-M. Kermarrec, L. Marchal, and E. Riviere. VoroNet: A scalable object network based on Voronoi tessellations. In *Proc. IEEE Intl. Parallel and Distributed Computing Symp. (21st IPDPS'07)*, pages 1–10, Long Beach, California, USA, Mar. 2007. IEEE Computer Society (Los Alamitos, CA).

[11] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig. NATBLASTER: Establishing TCP connections between hosts behind NATs. Apr. 2005.

[12] R. Bless, C. Huebsch, S. Mies, and O. Waldhorst. The Underlay Abstraction in the Spontaneous Virtual Networks (SpoVNet) Architecture. *Next Generation Internet Networks, 2008. NGI 2008*, apr. 2008.

[13] T. Bova and T. Krivoruchka. Reliable UDP Protocol. Internet-draft, Internet Engineering Task Force, 1999.

[14] R. Braden. Requirements for Internet Hosts - Communication Layers. Technical Report 1122, Oct. 1989. Updated by RFCs 1349, 4379, 5884, 6093, 6298.

[15] E. Buyukkaya and M. Abdallah. Data Management in Voronoi-Based P2P Gaming. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 1050 –1053, Jan. 2008.

[16] E. Buyukkaya and M. Abdallah. Efficient triangulation for P2P networked virtual environments. *Multimedia Tools Appl.*, 45:291–312, Oct. 2009.

[17] CACE Technologies. WinPcap. published in the WWW at http://www.winpcap.org.

[18] M. Castro, P. Druschel, A. J. Ganesh, A. I. T. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proceedings of the 5th ACM Symposium on Operating System Design and Implementation (OSDI-02)*, Operating Systems Review, pages 299–314, New York, Dec. 9–11 2002. ACM Press.

[19] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.

[20] C. Cramer and T. Fuhrmann. Bootstrapping chord in ad hoc networks: Not going anywhere for a while. In *Proceedings of the 3rd IEEE International Workshop on Mobile Peer-to-Peer Computing*, Pisa, Italy, March 2006.

[21] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*. Springer„ 2003.

[22] D. Damsker. "local control network and internetwork iso-osi refernece model". *Power Apparatus and Systems, IEEE Transactions on*, PAS-102(5):1202–1210, May 1983.

[23] T. Dierks and C. Allen. RFC 2246: The TLS Protocol Version 1.0. Technical report, Jan. 1999.

[24] J. R. Douceur. The Sybil Attack. In P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, editors, *IPTPS '01: Revised Papers from the First International Workshop*

on Peer-to-Peer Systems, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260, London, UK, 2002. Springer.

[25] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631, IETF, 1994.

[26] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704.

[27] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-Peer Communication Across Network Address Translators. In *USENIX Annual Technical Conference*, pages 179–192, 2005.

[28] M. Ghaffari, B. Hariri, and S. Shirmohammadi. A Delaunay Triangulation Architecture Supporting Churn and User Mobility in MMVEs. In *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '09, pages 61–66, New York, NY, USA, 2009. ACM.

[29] L. Gong. JXTA: A Network Programming Environment. *IEEE Internet Computing*, 5:88–95, 2001.

[30] S. Guha. STUNT – Simple Traversal of UDP Through NATs and TCP too. Technical report, Network Working Group, December 2004. Work in progress.

[31] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh. NAT Behavioral Requirements for TCP, Oct. 2008.

[32] S. Guha and P. Francis. Characterization and Measurement of TCP Traversal through NATs and Firewalls. *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005.

[33] S. Guha, Y. Takeda, and P. Francis. NUTSS: a SIP-based approach to UDP and TCP network connectivity. In *FDNA '04: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 43–48, New York, NY, USA, 2004. ACM.

[34] S. Hätönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo. An Experimental Study of Home Gateway Characteristics. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 260–266, New York, NY, USA, 2010. ACM.

[35] D. Hildebrandt, L. Bischofs, and W. Hasselbring. RealPeer–A Framework for Simulation-Based Development of Peer-to-Peer Systems. In *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, PDP '07, pages 490–497, Washington, DC, USA, 2007. IEEE Computer Society.

[36] S. Holzapfel, S. Schuster, and T. Weis. VoroStore - A Secure and Reliable Data Storage for Peer-to-Peer-Based MMVEs. In *IEEE 11th International Conference on Computer and Information Technology (CIT 2011)*, September 2011.

[37] S. Holzapfel, A. Wacker, T. Weis, and M. Wander. An Architecture for Complex Peer-to-Peer Systems. In *Proceedings of the 4th IEEE International Workshop on Digital Entertainment, Networked Virtual Environments, and Creative Technology, DENVECT 2012, held in conjunction with the IEEE Consumer Communications and Networking Conference, CCNC 2012*, Las Vegas, Nevada, USA, January 2012.

[38] S. Holzapfel, M. Wander, A. Wacker, L. Schwittmann, and T. Weis. A New Protocol to Determine the NAT Characteristics of a Host. In *Proceedings of 25th IEEE International Parallel Distributed Processing Symposium, International Workshop on Hot Topics in Peer-to-Peer Systems (HOTP2P)*, Anchorage, Alaska, USA, May 2011.

[39] S. Holzapfel, M. Wander, A. Wacker, and T. Weis. SYNI - TCP Hole Punching Based on SYN Injection. In *Proceedings of the The 10th IEEE International Symposium on Network Computing and Applications (IEEE NCA11)*, Cambridge, MA, USA, August 2011.

[40] S.-Y. Hu, S.-C. Chang, and J.-R. Jiang. Voronoi State Management for Peer-to-Peer Massively Multiplayer Online Games. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 1134–1138, 2008.

[41] S.-Y. Hu, J.-F. Chen, and T.-H. Chen. VON: A Scalable Peer-to-Peer Network for Virtual Environments. *Network, IEEE*, 20(4):22 – 31, July 2006.

[42] R. Jacob, S. Ritscher, C. Scheideler, and S. Schmid. A Self-stabilizing and Local Delaunay Graph Construction. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, ISAAC '09, pages 771–780, Berlin, Heidelberg, 2009. Springer-Verlag.

[43] J.-R. Jiang, Y.-L. Huang, and S.-Y. Hu. Scalable aoi-cast for peer-to-peer networked virtual environments. In *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems Workshops*, ICDCSW '08, pages 447–452, Washington, DC, USA, 2008. IEEE Computer Society.

[44] K. Jünemann, P. Andelfinger, and H. Hartenstein. Towards a Basic DHT Service: Analyzing Network Characteristics of a Widely Deployed DHT. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, pages 1 –7, 31 2011-aug. 4 2011.

[45] M. Knoll, M. Helling, A. Wacker, S. Holzapfel, and T. Weis. Boostrapping Peer-to-Peer Systems using IRC. In *5th International Workshop on Collaborative Peer-to-Peer Systems (COPS2009)*, Groningen, The Netherlands, June 29th - July 1st 2009.

[46] M. Knoll, A. Wacker, G. Schiele, and T. Weis. Bootstrapping in Peer-to-Peer Systems. In *14th IEEE International Conference on Parallel and Distributed Systems (ICPADS'08)*, Melbourne, Australia, 2008.

[47] M. Knoll and T. Weis. A P2P-Framework for Context-based Information. In *1st International Workshop on Requirements and Solutions for Pervasive Software Infrastructures (RSPSI) at Pervasive 2006*, Dublin, Ireland, May 2006.

[48] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen, and J. Vuori. P2PRealm – Peer-to-Peer Network Simulator. In *Proc. 11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks, 2006*, pages 93–99, 2006.

[49] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[50] P. G. López, C. Pairot, R. Mondéjar, J. P. Ahulló, H. Tejedor, and R. Rallo. PlanetSim: A New Overlay Network Simulation Framework. In *SEM*, volume 3437, pages 123–136. Springer, 2004.

[51] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005.

[52] D. MacDonald and B. Lowekamp. NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN). RFC 5780, IETF, May 2010.

[53] L. Makinen and J. Nurminen. Measurements on the feasibility of tcp nat traversal in cellular networks. In *Next Generation Internet Networks, 2008. NGI 2008*, pages 261 –267, april 2008.

[54] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, pages 99 –100, Sept. 2009.

[55] M. Mordacchini, L. Ricci, L. Ferrucci, M. Albano, and R. Baraglia. Hivory: Range Queries on Hierarchical Voronoi Overlays. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1 –10, Aug. 2010.

[56] A. Mueller, N. S. Evans, C. Grothoff, and S. Kamkar. Autonomous NAT Traversal. In *10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P'10)*, Delft, The Netherlands, 2010. IEEE, IEEE.

[57] A. Muller, G. Carle, and A. Klenk. Behavior and classification of nat devices and implications for nat traversal. *Network, IEEE*, 22(5):14 –19, september-october 2008.

[58] A. Muller, A. Klenk, and G. Carle. Ants - a framework for knowledge based nat traversal. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1 –6, 30 2009-dec. 4 2009.

[59] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, 1998.

[60] J. Postel. User Datagram Protocol. Technical Report 768, Aug. 1980.

[61] J. Postel. Internet Control Message Protocol. Technical Report 792, Sept. 1981. Updated by RFCs 950, 4884.

[62] J. Postel. Internet Protocol. Technical Report 791, Sept. 1981. Updated by RFC 1349.

[63] J. Postel. Transmission Control Protocol. RFC 793, IETF, September 1981.

[64] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 161–172, San Diego, CA, USA, 2001. ACM Press.

[65] L. Ricci and L. Genovali. State Management in Distributed Virtual Environments: A Voronoi Base Approach. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010 International Congress on*, pages 881 – 887, Oct. 2010.

[66] L. Ricci and A. Salvadori. Nomad: Virtual Environments on P2P Voronoi Overlays. In R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, volume 4806 of *Lecture Notes in Computer Science*, pages 911–920. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-76890-6_17.

[67] A. Rodriguez, C. Killian, S. Bhat, D. Kostic, and A. Vahdat. MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks. In *In NSDI*, pages 267–280, 2004.

[68] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389, IETF, October 2008.

[69] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489, IETF, 2003.

[70] R. Roverso, S. El-Ansary, and S. Haridi. Natcracker: Nat combinations matter. In *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, pages 1 –7, aug. 2009.

[71] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[72] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th ACM SOSP*, Lake Louise, Alberta, Canada, October 2001.

[73] M. Saternus. *Gears4Net - Ein Programmiermodell für asynchrone Kommunikation in P2P-Systemen*. PhD thesis, University of Duisburg-Essen, 2012.

[74] M. Saternus, M. Knoll, F. Dürr, and T. Weis. Symstry: Ein P2P-System für Ortsbezogene Anwendungen. In *Proceedings of 15. ITG/GI - Fachtagung (KiVS 2007)*, pages 99–104. VDE-Verlag, Februar 2007.

[75] M. Saternus, T. Weis, S. Holzapfel, and A. Wacker. Gears4Net - An Asynchronous Programming Model. In *International Conference on Parallel Processing Workshops, 2009. ICPPW '09.*, Sept. 2009.

[76] B. Schneier. *Applied cryptography - protocols, algorithms, and source code in C (2. ed.)*. Wiley, 1996.

[77] S. Schuster, A. Wacker, and T. Weis. Fighting Cheating in P2P-based MMVEs with Disjoint Path Routing. *Electronic Communications of the EASST*, 17, 2009.

[78] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations, 1999.

[79] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan. Middlebox communication architecture and framework. RFC 3303 (Informational), Aug. 2002.

[80] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, San Diego, California, United States, 2001. ACM.

[81] A. Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 2002.

[82] E. Tanin, A. Harwood, H. Samet, S. Nutanong, and M. T. Truong. A Serverless 3D World. In *GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 157–165, New York, NY, USA, 2004. ACM.

[83] P.-Y. Tarng, K.-T. Chen, and P. Huang. An analysis of wow players' game hours. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '08, pages 47–52, New York, NY, USA, 2008. ACM.

[84] UPnP Forum. UPnP Specifications. Published on the WWW at `http://upnp.org/sdcps-and-certification/standards/`. (retrieved on 2012-06-18).

[85] P. Urbán, X. Défago, and A. Schiper. Neko: A Single Environment to Simulate and Prototype Distributed Algorithms. In *Proceedings of the 15th Int'l Conf. on Information Networking (ICOIN-15)*, Beppu City, Japan, 2001.

[86] A. Wacker, G. Schiele, S. Holzapfel, and T. Weis. A NAT Traversal Mechanism for Peer-To-Peer Networks. In *P2P '08: Proceedings of the Eighth IEEE International Conference on Peer-to-Peer Computing (P2P'08)*, pages 81–83, Aachen, Germany, Sept. 2008. IEEE.

[87] M. Wander, S. Holzapfel, A. Wacker, and T. Weis. NTALG – TCP NAT Traversal with Application-Level Gateways. In *Proceedings of the 9th IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, Nevada, USA, January 2012.

[88] Y. Wang, Z. Lu, and J. Gu. Research on Symmetric NAT Traversal in P2P Applications. In *Proceedings of the International Multi-Conference on Computing in the Global Information Technology*, pages 59–, Washington, DC, USA, 2006. IEEE Computer Society.

[89] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

Liste der Publikationen

1. Arno Wacker, Gregor Schiele, Sebastian Holzapfel, and Torben Weis
   (Demo) A NAT Traversal Mechanism for Peer-To-Peer Networks
   Proceedings of the 8th International Conference on Peer-to-Peer Computing, IEEE P2P08, Aachen, Germany, September 2008.

2. Torben Weis, Arno Wacker, Sebastian Schuster, Sebastian Holzapfel
   Towards Logical Clocks in P2P-based MMVEs
   Proceedings of the 1st International Workshop on Concepts of Massively Multiuser Virtual Environments, CoMMVE09, organized at the 16th ITG/GI Fachtagung - Kommunikation in Verteilten Sytemen 2009, KiVS 2009, Kassel, March 2-6, 2009

3. Mirko Knoll, Matthias Helling, Arno Wacker, Sebastian Holzapfel and Torben Weis
   Bootstrapping Peer-to-Peer Systems Using IRC
   Proceedings of the 5th International Workshop on Collaborative Peer-to-Peer Systems, COPS09, organized at the 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE 2009, Groningen, The Netherlands, June 29th - July 1st, 2009

4. Martin Saternus, Torben Weis, Sebastian Holzapfel, Arno Wacker
   Gears4Net - an Asynchronous Programming Model
   Proceedings of the Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2), organized at the 38th International Conference on Parallel Processing 2009 (ICPP09), Vienna, Austria, Sept. 22-25, 2009

5. Sebastian Holzapfel, Matthäus Wander, Arno Wacker, Lorenz Schwittmann, and Torben Weis
   A New Protocol to Determine the NAT Characteristics of a Host
   8th International Workshop on Hot Topics in Peer-to-Peer Systems, HotP2P

2011, held in conjunction with the 25th IEEE International Parallel & Distributed Processing Symposium, IPDPS 2011, Anchorage, Alaska, USA, May 16-20, 2011.

6. Helge Parzyjegla, Arnd Schröter, Enrico Seib, Sebastian Holzapfel, Matthäus Wander, Jan Richling, Arno Wacker, Hans-Ulrich Heiß, Gero Mühl and Torben Weis
   Model-Driven Development of Self-Organizing Control Applications
   Chapter 2.1, in Organic Computing - A Paradigm Shift for Complex Systems, Editor: Christian Müller-Schloer and Hartmut Schmeck and Theo Ungerer, Birkäuser Verlag, Basel, Switzerland, June 2011.

7. Sebastian Holzapfel, Sebastian Schuster, and Torben Weis
   VoroStore – A Secure and Reliable Data Storagefor Peer-to-Peer-based MMVEs
   Proceedings of the International Workshop on Resource Discovery Mechanisms for P2P Systems, P2P-RDM 2011, held in conjunction with the 11th IEEE International Conference on Computer and Information Technology, CIT 2011, August, 31st – September, 2nd 2011 in Pafos (Paphos), Cyprus.

8. Sebastian Holzapfel, Matthäus Wander, Arno Wacker, and Torben Weis
   SYNI – TCP Hole Punching Based on SYN Injection
   Proceedings of the The 10th IEEE International Symposium on Network Computing and Applications, IEEE NCA11, Cambridge, MA, USA August 25 - 27, 2011

9. Sebastian Holzapfel, Arno Wacker, Torben Weis, and Matthäus Wander
   An Architecture for Complex Peer-to-Peer Systems
   Proceedings of the 4th IEEE International Workshop on Digital Entertainment, Networked Virtual Environments, and Creative Technology, DENVECT 2012, held in conjunction with the IEEE Consumer Communications and Networking Conference, CCNC 2012, Las Vegas, Nevada, USA January 14 - 17, 2012.

10. Matthäus Wander, Sebastian Holzapfel, Arno Wacker, and Torben Weis
    NTALG – TCP NAT Traversal with Application-Level Gateways (Demonstration)
    9th IEEE Consumer Communications and Networking Conference, CCNC 2012, Las Vegas, Nevada, USA, January 14 - 17, 2012