# Netzwerkdienste für Massively Multiplayer Online Games

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Dipl.-Inf. Tonio Triebel
aus
Freiburg

Mannheim, 2014

| | |
|---|---|
| Dekan: | Professor Dr. Heinz Jürgen Müller, Universität Mannheim |
| Referent: | Professor Dr. -Ing. Wolfgang Effelsberg, Universität Mannheim |
| Korreferent: | Professor Dr. Thorsten Strufe, Technische Universität Dresden |

Tag der mündlichen Prüfung: 10. Juli 2014

# Danksagung

An dieser Stelle möchte ich mich bei den Menschen bedanken, die mich bei der Erstellung der Dissertation unterstützt haben. Vor allem möchte ich mich bei Professor Effelsberg für die hervorragende Betreuung und für seine unendliche Geduld bedanken. Sodann bei Ares und Alexa, die auch in schwierigen Phasen an meiner Seite standen und somit erst die Fertigstellung der Arbeit ermöglicht haben. Ebenso geht mein Dank an Benjamin Guthier und Max Lehn, die immer hilfsbereit waren und mit deren Zusammenarbeit der Großteil der Pulikationen entstanden ist.

# Abstract

In this theses we present the design and implementation of three different network services for massively multiplayer online games. The first service is an approach for an integrated voice communication. The design is based on a hybrid architecture using a server-based signaling and peer-to-peer data transmission. This technique allows to relate speech processing to game events throuhgt the server but does not consume centralized ressources for the audio transmission. The second service is an approach for an overlay network structure. Aiming for high demands on responsiveness and scalability, we use a fully distributed peer-to-peer network with a dynamic connection scheme. By using a publish/subscribe system for message dissemination user interests can be handled efficiently. A Geocast algorithm allows information distribution to arbitrary regions of the virtual world. The third service can be used to benchmark peer-to-peer gaming overlays. It contains the design of the methodology, the definition of metrics, the generation of realistic gaming workload and the implementation of a complete evaluation platform. All three services are using the game Planet PI4 as a prototype for a distributed massively multiplayer first person shoter online game. Its modular design allows a versatile application of the game. Network relevant game events can be created by real players or simulated by simple mobility models or complex artificial intelligence players. Planet PI4 also servers as a possibility to see and feel the quality of an implemented service in a realtime action game.

# Zusammenfassung

In dieser Arbeit präsentieren wir das Design und die Umsetzung von drei Netzwerkdiensten für Massively Multiplayer Online Games. Der erste Dienst stellt eine integrierte Sprachkommunikation für Online-Spiele dar. Dabei basiert das Design auf einer hybriden Architektur, die für das Aushandeln der Verbindungen einen zentralen Server verwendet, die Audiodaten jedoch von Peer zu Peer übertägt. Dieses Vorgehen ermöglicht es, die Verbindungen basierend auf Spielereignissen aufzubauen, ohne dabei zentrale Ressourcen für das Übertragen der Audioströme zu verbrauchen. Der zweite Dienst ist ein Ansatz für eine Overlay-Netzwerkstruktur. Das Hauptziel bei der Entwicklung war es, hohe Echtzeit- und Skalierbarkeitsanforderungen zu erfüllen. Daher wird ein unstrukturiertes Peer-to-Peer-Netzwerk mit einem dynamischen Verbindungsschema verwendet. Durch die Verwendung eines Publish-/Subscribesystems für die Verbreitung der Nachrichten können alle Anforderungen effizient erfüllt werden. Ein Geocastalgorithmus ermöglicht zusätzlich das Verbreiten von Nachrichten an beliebige Regionen einer virtuellen Welt. Der dritte Dienst kann für den Leistungsvergleich von Peer-to-Peer-Gaming-Overlays eingesetzt werden. Er beeinhaltet das Festlegen der Methodik, die Definition von Metriken, die Generierung von realistischer Last und die Implementierung einer vollständigen Evaluationsplattform. Alle drei Dienste verwenden Planet PI4 als Prototyp einer verteilten Massively Multiplayer First Person Shooter Spieleanwendung. Dabei ermöglicht die modualre Architektur eine vielseitige Anwendbarkeit des Spiels. Netzwerk-relevante Ereignisse können sowohl von realen Spielern als auch von einfachen Bewegungsmodellen oder komplexen KI-Spielern erzeugt werden. Darüber hinaus dient Planet PI4 als Möglichkeit, die Qualität eines Netzwerkdienstes direkt anhand eines Echtzeit-Actionspiels zu erleben.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1

# Einleitung

*„Der Mensch spielt nur, wo er in voller Bedeutung des Wortes Mensch ist, und er ist nur da ganz Mensch, wo er spielt.“*

Fr. Schiller (1793) [48]

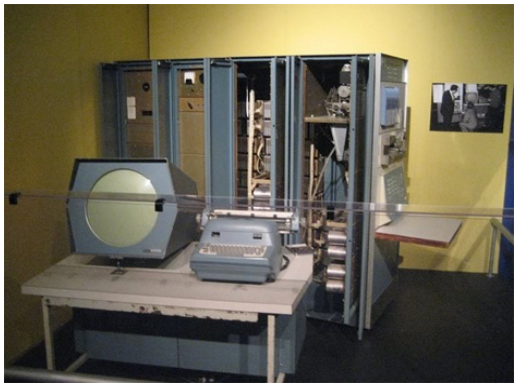
Spiele begleiten den Menschen seit seiner Entstehung. Sie sind der grundlegende Mechanismus zum Erlernen der Abläufe des Lebens. Aufgrund der Evolution des Menschen und dem damit verbundenen Wandel der Abläufe verändert sich ebenso die Art und Weise des Spielens. So standen bei frühen Spielformen wie beispielsweise dem *Verstecken* oder *Fangen* das Erlernen von Überlebensstrategien im Vordergrund. Diese primitiven Verhaltensmuster sind zwar bis heute erhalten geblieben, jedoch sind durch die veränderten Anforderungen des Lebens und durch technische Entwicklungen viele neue Formen des Spielens hinzugekommen. Dabei haben vor allem die rasante Entwicklung der Computerhardware und die zunehmende Vernetzung der Welt zu einem Fortschreiten der Spiele in Komplexität, Umfang und Ausdehnung geführt. So können heutzutage nahezu alle motorischen Bewegungsabläufe wie Sport, das Bewegen von Fahrzeugen oder das Erlernen von Instrumenten bis zu einem gewissen Grad durch Computersimulationen erlernt werden. Zusätzlich können auch kognitive und sogar soziale Fähigkeiten ausgebildet werden. Um die dabei immer komplexer werdenden Spielszenarien darstellen zu können, müssen stets neue Techniken entwickelt werden. Daher stehen Spiele auch immer

öfter im Fokus der Wissenschaft. Es gilt, hoch-skalierbare verteilte Systeme zu entwickeln, die von tausenden Spielern als gemeinsame Simulationsumgebung genutzt werden können. Dabei entstehen virtuelle Welten, in denen sich die Spieler bewegen und miteinander interagieren können. Die größte Herausforderung bei einer derartigen virtuellen Welt stellt das Verwalten des Weltzustandes dar, da die Aktionen der Spieler meist in Echtzeit synchronisiert werden müssen. Treten dabei Fehler auf, kann das Lernergebnis oder der Spaß am Spielen schnell beeinträchtigt werden. Daher ist das Ziel dieser Arbeit, Netzwerkmechanismen zu entwickeln, die für die Synchronisierung von hochskalierbaren virtuellen Welten eingesetzt werden können, und eine Möglichkeit zu finden, die Qualität unterschiedlicher Mechanismen miteinander zu vergleichen. Dazu wird zunächst die Bedeutung der Thematik durch technische, wirtschaftliche und gesellschaftliche Aspekte motiviert. Im Anschluss daran werden die notwendigen Grundlagen für eine Umsetzung eingeführt. Am Ende des Grundlagenkapitels werden dann die konkreten Forschungsfragen gestellt, die im weiteren Verlauf der Arbeit behandelt werden.

## 1.1 Motivation

*„Ernst ist Nichtspiel und nichts anderes. Der Bedeutungsinhalt von Spiel dagegen ist mit Nichternst keineswegs definiert oder erschöpft: Spiel ist etwas Eigenes“* Johan Huizinga (1939) [26].

Dieser Gedanke des niederländischen Kulturhistorikers Johan Huizinga widersprach bereits 1939 der gängigen Meinung, dass Spiel durch die Abwesenheit von Ernst definiert werden kann. Diese heute noch weit verbreitete Meinung hat zur Folge, dass die Bedeutung der Spieleindustrie von vielen Menschen unterschätzt wird. Spielen gilt für sie als reines Freizeitvergnügen und hat somit nur einen geringen Stellenwert. Welchen gewaltigen Wandel und damit verbundenen Zuwachs an Bedeutung Spiele jedoch in den vergangenen Jahrzehnten vollzogen haben, soll im Folgenden aus technischer, wirtschaftlicher und gesellschaftlicher Sicht verdeutlicht werden. Alle drei Sichtweisen beinhalten motivierende Argumente, sich langfristig wissenschaftlich mit der technischen Umsetzung von Spielen auseinanderzusetzen.



(a) Das weltweit erste Computerspiel Cathode Ray Tube Amusement Device (CRTAD) aus dem Jahre 1948

(b) Das Spiel OXO nutze 1952 das erste grafische Display


(c) Homepong war die erste Spielekonsole, die für den Massenmarkt konzipiert wurde

(d) Spaceinvaders erzielte in den ersten zwei Jahren einen Umsatz von über 500 Millionen US Dollar.

Abbildung 1.1: Entwicklung der ersten Computerspiele.(Bildquellen: Wikipedia/Wikimedia)

### 1.1.1 Technische Entwicklung der Spiele

Um einen ersten Eindruck vom Wandel der Spiele zu bekommen, ist es hilfreich, sich den technischen Entwicklungsprozess der Spiele genauer anzuschauen. Der Ursprung der technischen Entwicklung von Spielen besteht in dem fundamentalen Wandel des Spielmediums. So waren Spiele zunächst immer Produkte der physischen Welt. Dies änderte sich jedoch durch die Entwicklung von Computern. Die Z3 von Konrad Zuse (1941) stellte die erste turingmächtige Maschine dar, die in der Lage war, beliebige Programme und Vorschriften automatisch auszuführen. In Verbindung mit entsprechenden Informationsverarbeitungs-

(a) Bild der LAN-Party *DreamHack 2004* bei der über 5000 Spieler zusammen kamen. Der aktuelle Weltrekord wurde 2007 bei einer Folgeveranstaltung erzielt, an der sich 10.455 Spieler beteiligten.

(b) Bei der *Schlacht von Asakai* bekämpften sich in EVE Online 2754 Spieler und vernichteten in einem mehrstündigen Gefecht virtuelle Güter im Wert ca. 70.000 Dollar

Abbildung 1.2: Zwei Beispiele für die dramatische Entwicklung, die Computerspiele in den letzten Jahren erfahren haben. (Bildquelle: Wikipedia)

möglichkeiten wie Eingabe, persistente Speicherung und Ausgabe von Daten war somit die Möglichkeit geschaffen, physische Prozesse virtuell zu simulieren. Bis zur ersten tatsächlichen Umsetzung eines funktionsfähigen Computerspiels dauerte es jedoch noch bis zum Jahre 1948. In dem U.S. Patent 2.455.992 wird das erste offizielle Computerspiel mit dem Titel *Cathode Ray Tube Amusement Device (CRTAD)* beschrieben (siehe Abbildung 1.1a). Die Inspiration für das Spiel stammte von einem Radardisplay aus dem zweiten Weltkrieg. Die Aufgabe des Spielers bestand darin, die Flugbahn und Geschwindigkeit eines sich bewegenden Punktes über verschiedene Drehknöpfe so auszurichten, dass er bestimmte Ziele trifft. Da es noch kein grafisches Display gab, wurden die Ziele auf eine Folie gemalt und auf dem CRT angebracht. Aufgrund der hohen Kosten für das Gerät erreichte das CRTAD jedoch nie Marktreife.

1952 entwickelte der Doktorand A. S. Douglas an der Universität von Cambridge das erste grafische Computerspiel. Er nutze einen EDSAC Vakuum-Röhren-Computer mit einem $35x16$ Bildpunkte CRT-Display. Douglas implementierte eine grafische Variante von *Tic-Tac-Toe* namens OXO (siehe Abbildung 1.1b). Auch heute kann man OXO noch auf einem EDSAC-Emulator

spielen. In den folgenden Jahren wurden weitere Spiele für Großrechner entwickelt. Dabei waren vor allem Studenten und Mitarbeiter von Universitäten und Forschungseinrichtungen an den Implementierungen beteiligt. So wurde 1958 am Brookhaven National Laboratory *Tennis for Two* von William Higinbotham vorgestellt. Es diente zur Unterhaltung der Besucher. 1961 wurde dann das Spiel *Spacewar* von den MIT-Studenten Martin Graetz, Steve Russell und Wayne Wiitanen's auf einem DEC PDP-1 Computer geschrieben. Der erste kommerzielle Erfolg wurde 1972 mit Atari's *Pong* erzielt. Etwa zur selben Zeit entwickelte der Programmierer und Hobbyhöhlenforscher Will Crowther das Spiel *Colossal Cave Adventure*. Der Name „Adventure" steht heute noch für eines der größten Spielegenres. Auf der Basis von *Colossal Cave Adventure* entwickelte 1978 Roy Trubshaw an der Essex University das erste Mehrspielerspiel *Multi-User Dungeon (MUD)*. Aufgrund der fehlenden Infrastruktur etablierten sich Netzwerkspiele jedoch erst Mitte der 1990er Jahre. Ende der 1970er bzw. Anfang der 1980er begann unterdessen die Ära der Homecomputer. So war mit dem C64 erstmals ein spielefähiger Computer dem Massenmarkt zugänglich.

Die nächste Evolutionsstufe wurde 1993 mit dem Erscheinen von *Doom* eingeleitet. Doom zeichnete sich nicht nur durch eine bahnbrechende Grafik und einen hochwertigen Sound aus, sondern es ermöglichte auch durch das *Binary Space Partitioning (BSP)* erstmals eine flüssige Darstellung von großen dreidimensionalen Szenen. Die wichtigste Neuerung war jedoch die Bereitstellung eines Netzwerkinterfaces, wodurch die Spieler zum ersten Mal in einem lokalen Netzwerk mit bis zu drei weiteren Spielern gegeneinander spielen konnten. Diese Funktion löste einen regelrechten Hype unter den Spielern aus und sorgte für eine neue Form des gesellschaftlichen Spielens, die Local Area Network (LAN)-Parties. Bei diesen Events trafen sich zunächst meist nur befreundete Spieler in einem kleinen privaten Kreis. Später wurden daraus immer größere öffentliche Veranstaltungen, bei denen sich zum Teil hunderte von Spielern in Sporthallen oder Konferenzräumen versammelten, um gemeinsam zu spielen (siehe Abbildung 1.2a).

Aus technischer Sicht wurde der aktuell letzte Paradigmenwechsel [1] von

[1] Mobile Games können als weiterer Technologiesprung angesehen werden. Sie nutzen jedoch lediglich eine andere Zugangstechnologie und erfordern keine neuartige Netzwerkar-

Origin Systems durch das Spiel *Ultima Online* vollzogen. Ultima Online war das erste Spiel, das es einer großen Menge von Spielern ermöglichte, nicht nur im lokalen Netzwerk, sondern über das globale Internet gemeinsam zu spielen. Der große Erfolg von Ultima Online begründete das Genre der *Massively Multiplayer Online Games (MMOG)*, das heute neben Shooter- und Sportspielen das größte Marktsegment der Spieleindustrie bildet. In aktuellen MMOGs wie beispielsweise *Eve Online* können zehntausende Spieler gleichzeitig in einer einzigen Welt miteinander spielen. Regelmäßig kommt es dabei zu gewaltigen Raumschlachten, in denen sich tausende Spieler gegenseitig bekämpfen (siehe Abbildung 1.2b). Um einen realistischen Ablauf einer derartigen Raumschlacht zu ermöglichen, müssen sämtliche Bewegungen und Interaktionen aller Spieler in Echtzeit an einen dedizierten Server übertragen und in die virtuelle Welt integriert werden. Dieser Weltzustand muss dann wiederum mit den Spielern synchronisiert werden. Dass bei normalem Spielbetrieb dabei keine Verzögerungen spürbar sind, zeugt von der hohen Leistungsfähigkeit der aktuellen Spielanwendungen. Dennoch besteht nach wie vor eine Grenze, die nicht überschritten werden kann. Die Ursache dafür besteht darin, dass sämtliche MMOGs derzeit einen dedizierten Server für die Verwaltung ihrer virtuellen Welten nutzen. Diese haben die inhärente Eigenschaft, dass sie eine Obergrenze für die Anzahl der Spieler haben, die durch die Leistungsfähigkeit der Hardware bestimmt wird. Die Möglichkeit einer globalen virtuellen Welt, die potentiell von alle Menschen gleichzeitig genutzt werden kann (und eventuell auch als intuitives Interface für das Internet dienen könnte), kann mit einer derartigen Client-Server-Architektur in absehbarer Zeit nicht realisiert werden. Daher gilt es, Möglichkeiten zu finden, virtuelle Welten auch dezentral zu verwalten. Die dezentrale Verwaltung von Daten und Algorithmen ist die Kerndisziplin der *Verteilte-Systeme-Forschung.*

### 1.1.2 Zuwachs der wirtschaftlichen Bedeutung

Mit der technischen Weiterentwicklung der Computerspiele in den vergangenen Jahrzehnten ist auch ihr kommerzieller Erfolg verbunden. Dadurch ist die Spieleindustrie zu einem bedeutenden Wirtschaftsfaktor geworden, der viele

---

chitektur

Arbeitsplätze bereitstellt. Den ersten Durchbruch brachten dabei zunächst die Arkade-Spielautomaten, die in Spielhallen aufgestellt wurden. Bereits die ersten primitiven Pong-Automaten wurden 2500 (1973) bzw. 8000 (1974) mal verkauft. Mit dem Titel *Space Invaders* (siehe Abbildung 1.1d) schaffte dann die japanische Firma Taito 1978 den endgültigen Durchbruch. In den ersten zwei Jahren erzielte sie einen Gewinn von 500 Millionen US Dollar. Der Film *Star Wars* erzielte im Vergleich dazu 1977 einen Gewinn von 175 Millionen US Dollar. Dabei setzt sich das rasante Wachstum der Spieleindustrie bis heute fort, und der weltweite Gesamtumsatz übertraf 2008 erstmals den Umsatz der Kino- oder Musikindustrie. 2012 wurden weltweit mit Konsolen und Computerspielen ca. 65 Milliarden Dollar umgesetzt. Welche Bedeutung dieser Zuwachs hat, ist besonders an dem erfolgreichsten aller MMOGs, *World of Warcraft (WoW)* der Firma Blizzard zu erkennen. Mit bis zu 12 Millionen aktiven Abonnenten übertrifft es alle anderen MMOGs. Dabei ist die Nutzung der WoW-Server mit monatlichen Gebühren von über 10 Euro verbunden. D.h., allein WoW erzeugt einen jährlichen Umsatz von über einer Milliarde Euro. Das Ausmaß von WoW lässt sich am besten an einer Beschreibung des Blizzard-Managers Frank Pearce erkennen.

*Über die Infrastruktur hinter World of Warcraft hat Blizzard-Manager Frank Pearce auf der Game Developers Conference (GDC) in Austin gesprochen und Zahlen genannt. Sein Team hat bislang rund 180.000 Bugs in dem Onlinerollenspiel gefixt. Am Code arbeiten gerade mal 32 Programmierer, die in Teams für die Servertechnologie, die Engine oder die Benutzeroberfläche zuständig sind. 5,5 Millionen Zeilen Quellcode seien bislang entstanden. Des Weiteren arbeiteten an dem Onlinerollenspiel 30 leitende Angestellte. Insgesamt gibt es 37 Designer und 51 Grafiker. 123 Mitarbeiter sind bei Blizzard allein für Zwischensequenzen, Intros und andere Filme zuständig. 245 Menschen seien für die Übersetzung, Betreuung der bislang zehn Sprachversionen und weitere Qualitätssicherungsaufgaben zuständig. Blizzard beschäftige weltweit rund 340 Mitarbeiter, die sich um die Buchhaltung kümmern, sowie 2.056 Gamemaster für den Kundendienst im Spiel und 66 Communitymitarbeiter insbesondere für die Foren. Mit Marketing, PR und weiteren Abteilungen würden derzeit rund 4.600 Menschen für World of Warcraft arbeiten. Auch die Zahlen über die verwendete Technik sind imposant: Damit World of Warcraft läuft, sind bei Blizzard*

*13.250 Blade-Server im Einsatz sowie 75.000 CPUs und 112,5 TByte RAM für die Betreuung der Rechenzentren in Paris, Texas oder Seoul sind gerade mal 68 Mitarbeiter zuständig. Blizzard selbst hat laut Pearce insgesamt etwa 20.000 Computer im Einsatz und rund 1,3 Petabyte an Daten seien bislang entstanden.* [2]

Mitarbeiterzahlen und Hardwarebedarf sind zwar nur Begleiterscheinungen, aber sie verdeutlichen den enormen Zuwachs an wirtschaftlicher Bedeutung. Dieser hat seinen Ursprung in der großen Nachfrage des Marktes, da die Menschen leidenschaftliche Spieler und zugleich an technischen Innovationen interessiert sind. Dadurch sind sie auch bereit, viel Zeit in Spiele zu investieren.

### 1.1.3 Wandel der gesellschaftlichen Stellung

Die rasante technische und wirtschaftliche Entwicklung der Spiele hatte ebenso einen Wandel der gesellschaftlichen Stellung der Spiele zur Folge. Dieser hat sich jedoch deutlich langsamer vollzogen. Als Mitte der 1980er erstmals Computer dem Massenmarkt zugänglich gemacht wurden, standen viele Menschen der Entwicklung mit Skepsis gegenüber. Insbesondere, da die Bedienbarkeit der ersten Betriebssysteme noch nicht sehr intuitiv und benutzerorientiert war. Kinder und Jugendliche dagegen waren schnell von der neuen, bunten Welt begeistert. Diese Generation entwickelte mit der Zeit eine eigene Spielkultur. So wurden aus einfachen Spielen sportliche Wettkämpfe, in denen schnell die ersten Meisterschaften ausgetragen wurden. Heutige *E-Sports*-Turniere haben sich besonders im asiatischen Raum zu medialen Großereignissen entwickelt, bei denen die Champions wie Rockstars gefeiert und mit beträchtlichen Preisgeldern belohnt werden. So wurde beispielsweise für den Weltmeistertitel 2012 in *Starcraft 2* eine Prämie von 100.000 US Dollar gezahlt.

Neben diesem Wachstum an Intensität hat sich allmählich auch eine breitere Akzeptanz eingestellt. Durch die Vereinfachung der Bedienbarkeit und die Zeit des *Pervasive Computings*, in der in jeder Lebenslage ein Display zum Spielen verfügbar ist, gibt es mittlerweile keinerlei Berührungsängste mehr. D.h., die gesellschaftliche Akzeptanz für *Pervasive Gaming* ist gegeben.

[2] Die Beschreibung wurde dem Blog `http://www.golem.de/0909/69928.html` entnommen und gekürzt. Das Originalinterview entstammt der Game Developer Conference GDC 2009 in Austin

# Kapitel 2

# Grundlagen

## 2.1 Virtuelle Welten

„Virtuelle Welt“ ist der Oberbegriff für computergrafische Abbildungen von realen oder fiktiven Welten. Das kann beispielsweise das Cockpit eines Flugzeuges und dessen Umgebung in einem Flugsimulator oder die Fantasiewelt eines MMOG sein. Eine virtuelle Welt ist die Basis für nahezu jedes Compu-

terspiel. Eine besondere Herausforderung entsteht dabei, wenn diese Welt von mehreren Personen gleichzeitig genutzt wird, denn dann muss eine Vielzahl an Informationen zwischen den Teilnehmern ausgetauscht werden. So müssen Statusinformationen, wie beispielsweise die aktuelle Position oder das Aussehen eines virtuellen Charakters, und Ereignisse, die in der Welt geschehen, an andere übertragen werden. Zusätzlich sind Nachrichten für die Text- oder Sprachkommunikation notwendig. Jede Art von Information stellt dabei andere Anforderungen an das Netzwerk. So benötigt die Übertragung von Sprache eine hohe Bandbreite, das Synchronisieren von Echtzeitereignissen hingegen eine kurze Übertragungszeit. Um dennoch eine einzige, homogene virtuelle Welt zu ermöglichen, werden unterschiedliche Netzwerkdienste, -mechanismen, -protokolle und -architekturen eingesetzt. Die Bedeutung der einzelnen Komponenten und deren Zusammenspiel sollen in diesem Kapitel dargestellt werden. Dabei spielen die Qualitätskriterien, anhand derer die Leistung einzelner Komponenten bewertet werden kann, eine wichtige Rolle. Zuvor wird jedoch der allgemeine Aufbau einer virtuellen Welt und der darin enthaltenen Objekte diskutiert. Am Ende des Kapitels werden dann existierende Implementierungen vorgestellt und ungelöste Probleme aufgezeigt.

## 2.2 Aufbau einer virtuellen Welt

Die grafische Darstellung einer virtuellen Welt besteht aus einer dreidimensionalen Umgebung und den darin enthaltenen Objekten und Personen. Die Personen werden durch Avatare repräsentiert, die als Schnittstelle zwischen Spieler und Welt dienen. Avatare, Objekte und Umgebung bestehen jeweils aus zwei Komponenten, einem Modell (Mesh), das die Form definiert, und Texturen, die das Aussehen der Oberfläche bestimmen. Das Erzeugen der Meshes und Texturen ist Bestandteil eines Designprozesses und je nach Qualität mit großem Aufwand verbunden. Abbildung 2.1 zeigt die Oberflächentexturen eines Zwergenavatars, die zu einem Atlas zusammengefügt wurden. Zusammen mit dem Mesh-Modell, welches die 3D-Koordinaten aller Punkte des Avatars und zusätzlich Metadaten wie beispielsweise Normalenvektoren enthält (siehe Abbildung 2.2), kann mittels einer geeigneten Grafikbibliothek (3D-Engine) ein Bild des Avatars berechnet und angezeigt werden (siehe Abbildung 2.3).

Abbildung 2.1: Oberflächengrafiken eines Avatars und seiner Axt, die zu einem Textur-Atlas zusammengefügt wurden

Dafür stellen 3D-Engines Funktionen bereit, die abhängig von Beleuchtungs- und Kameraparametern eine konkrete Sicht auf die generierte 3D-Szene erzeugen können. Man unterteilt dabei in Implementierungen, die aus Sicht des Avatars angezeigt werden (First Person), und Welten, die aus einer Art Vogelperspektive betrachtet werden (Third Person). Um den Eindruck einer realen Welt mit großer Ausdehnung zu vermitteln, wird die virtuelle Szene von einem besonderen Objekt umschlossen, dass sich in der Unendlichkeit befindet. Das bedeutet, dass es außerhalb der Weltkoordinaten liegt und von Spielern, die sich in diese Richtung bewegen, niemals erreicht werden kann. Je nach Ausprägung der Form des umschließenden Objektes spricht man von einer Skybox (Würfel) oder einem Skydome (Kugel, Zylinder). Abbildung 2.4 zeigt Teile einer Skybox, die zur Veranschaulichung mit einem Abstand zueinander dargestellt werden. Das Terrain in der Mitte symbolisiert die gesamte Ausdehnung einer virtuellen Welt.

Umgebung, Objekte, Avatare und Skybox bilden den statischen Bestandteil einer Welt. Um Interaktionen und Bewegungen zu ermöglichen, wird zusätzlich eine kontinuierliche Funktion benötigt. Diese muss in der Lage sein, sowohl Nutzereingaben zu registrieren als auch den neuen Weltzustand grafisch auszugeben. Dazu wird nach der Initialisierung der statischen Bestandteile der Welt in eine zentrale Schleife gesprungen (main loop). In dieser Schleife werden

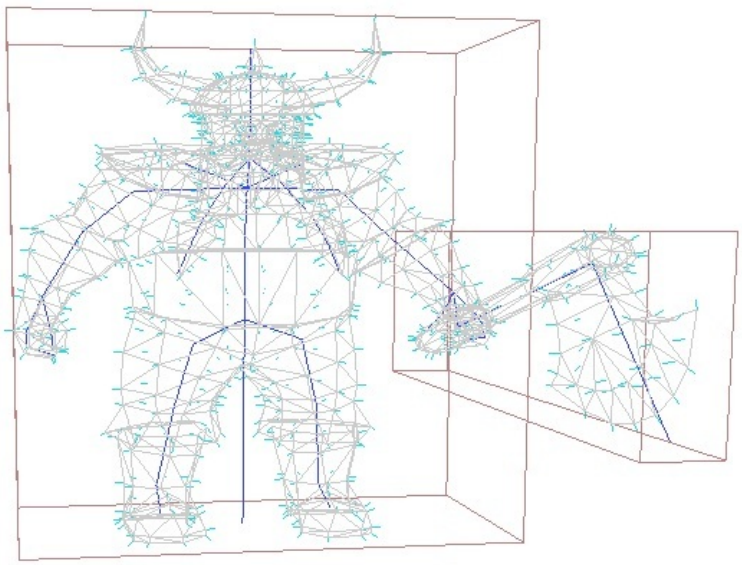


Abbildung 2.2: 3D-Mesh eines Zwergenavatars. Zusätzlich zu den 3D-Koordinaten werden Normalenvektoren und Boundingboxen angezeigt

Abbildung 2.3: Fertig berechneter Zwergenavatar

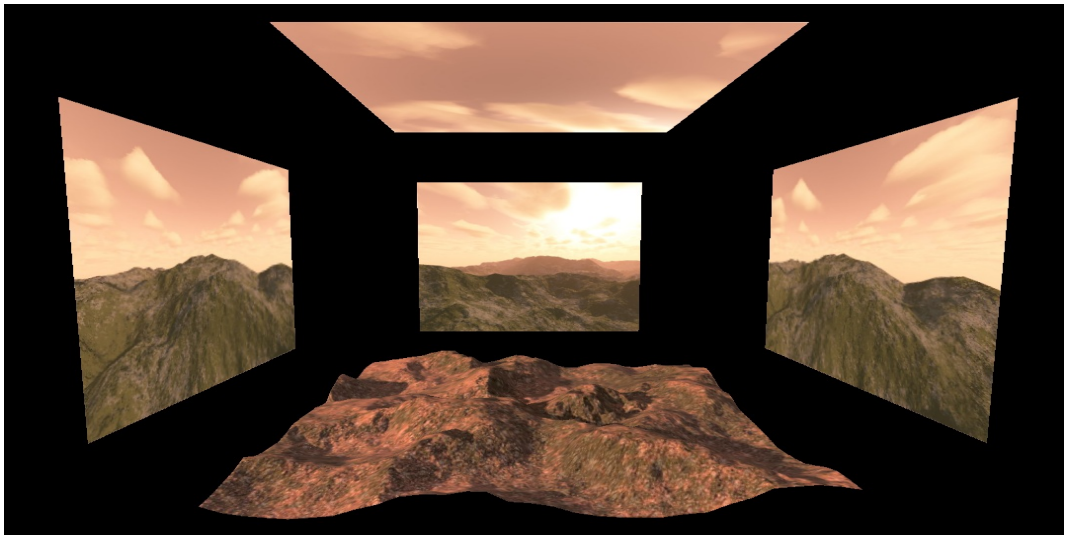


Abbildung 2.4: Teile einer Skybox, die außerhalb der virtuellen Welt liegen. Zusammenfügt erzeugen sie den Eindruck einer weiten Ausdehnung der virtuellen Welt.

dann Funktionen, die eine kontinuierliche Bearbeitung benötigen, wiederholend abgearbeitet. Da die Frequenz, in der die Schleife durchlaufen wird, die Interaktionsfähigkeit der Welt und die Aktualität der Szene bestimmt, wird sie meist ohne Taktvorgaben umgesetzt. D.h., die Durchlaufrate wird nur von der Leistungsfähigkeit der Hardware bestimmt. Je nach Echtzeitbedarf können alle weiteren kontinuierlichen Funktionen mit einer entsprechenden Frequenz ausgeführt werden. Um beispielsweise eine möglichst flüssige Darstellung von Bewegungen zu erreichen, wird die Aktualisierung der Grafik meist in jedem Durchlauf der main loop ausgeführt. Die Anzahl der erzeugten Bilder pro Sekunde wird als *Framerate* bezeichnet. Funktionen, die auf bestimmte Ereignisse reagieren, werden als Eventlistener-Funktionen bezeichnet. Sie bestehen aus einem Zustand, den sie beobachten, und Funktionen, die sie in Abhängigkeit von diesem Zustand ausführen. So werden beispielsweise Bewegungen über Tastatureingaben gesteuert. Ist der Zustand einer bestimmten Taste *„gedrückt“*, wird die Funktion für die entsprechende Bewegung ausgeführt. Zusätzlich zu Eventlistener-Funktionen und Grafikupdates nutzen verteilte virtuelle Welten die main loop zum Senden, Empfangen und Verarbeiten von Netzwerknachrichten.

## 2.3 Herausforderungen

Die Implementierung einer virtuellen Welt stellt eine synthetische Umgebung dar, in die sich ihre Nutzer hineinversetzen können. Ein bedeutendes Ziel bei der Entwicklung von virtuellen Welten ist es daher, den Effekt des Eintauchens bei den Anwendern bestmöglich zu stimulieren. Von diesem Effekt wird bei Simulationen die Realitätsnähe bzw. bei Spielen der Spielspaß direkt beeinflusst. Wie stark diese Eintauchfähigkeit (*immersiveness*) einer virtuellen Welt ist, lässt sich allerdings nicht direkt messen, da sie nicht nur von technischen, sondern auch von emotionalen Faktoren abhängt. Auch ein spannendes Buch kann den Leser in eine beschriebene Szene versetzen und ihn seine reale Umgebung vergessen lassen, ohne dabei überhaupt technische Hilfsmittel einzusetzen. Nicht-technische Mittel wie Atmosphäre, Spannung, Dramatik etc. werden ebenso in virtuellen Welten eingesetzt. Zusätzlich gilt es jedoch, die Repräsentation der virtuellen Welt so authentisch wie möglich zu gestalten. In verteilten virtuellen Welten beinhaltet das vor allem den Austausch von Netzwerknachrichten, da die Zustände der einzelnen Teilnehmer an alle anderen übermittelt werden müssen. Im Idealfall wird diese Synchronisation der Zustände derart realisiert, dass die verteilte Struktur des Systems dem Nutzer verborgen bleibt. Man bezeichnet diese Eigenschaft als *Transparenz*. In verteilten Systemen gibt es dabei unterschiedliche Teilkriterien, von denen die globale Transparenz abhängt. So ermöglicht beispielsweise eine Zugriffstransparenz den Zugriff auf lokale und entfernte Ressourcen unter Verwendung identischer Operatoren [17]. Allgemein lässt sich der abstrakte Begriff der globalen Transparenz durch die Erfüllung verschiedener konkreter Qualitätskriterien erreichen.

### 2.3.1 Qualitätskriterien

Qualitätskriterien dienen dazu, die komplexe Struktur eines Kommunikationssystems in einzelne Leistungsmerkmale zu unterteilen. Obwohl die einzelnen Kriterien in der Regel einem entsprechenden Netzwerkmechanismus zugeordnet werden können, können sie nicht direkt für eine Leistungsbewertung eingesetzt werden. Dazu werden Funktionen benötigt, die in der Lage sind, Eigenschaften explizit zu messen *(Metriken)*. Dennoch sind die Qualitätskriterien

von großer Bedeutung, da sie den Entwurf einer Architektur maßgeblich beeinflussen. So benötigt ein Netzwerk, bei dem Robustheit ein wichtiges Kriterium ist, redundante Ressourcen, um Ausfälle von Knoten kompensieren zu können. Metriken für Robustheit wären in diesem Fall die maximale Anzahl an Knotenverlusten, die kompensiert werden kann, oder die Verzögerung, die bei der Wiederherstellung verlorener Daten entsteht. Allgemein lassen sich Qualitätskriterien in technische Merkmale wie *Konsistenz, Korrektheit, Robustheit, Persistenz und Responsiveness* und in abgeleitete Merkmale wie *Skalierbarkeit, Fairness und Sicherheit* unterteilen. Diese Begriffe werden nachfolgend näher erläutert.

### Konsistenz

Konsistenz ist ein Begriff, der mit der Replikation von Daten verbunden ist. Er wird sowohl auf physikalischer Ebene (z.B. bei Prozessoren mit mehreren Speichern) als auch auf der logischen Ebene (z.B. Zustände in verteilten Systemen) verwendet.

In verteilten Spielen bezieht sich die Konsistenz auf den Zustände von Objekten der virtuellen Welt. Da jeder Spieler eine eigene Sicht der Welt verwaltet, müssen die Objekte repliziert werden. Wird der Zustand eines Objektes verändert, müssen alle Replikate synchronisiert werden. Dabei können unterschiedliche Fehler auftreten. So können Operationen durch ein Netzwerk verzögert werden oder verloren gehen. Welche Folgen daraus resultieren, hängt von dem verwendeten Konsistenzmodell eines System ab. Je nach Modell werden dabei unterschiedlich strenge Anforderungen an die Replikate der Objekte gestellt. Für verteilte Spiele sind die drei relevantesten Modelle:

**Strikte bzw. starke Konsistenz:** Das Modell fordert, dass jede Leseoperation eines Objektzustandes das exakte Ergebnis der letzten Schreiboperation liefert. D.h., der Zustand aller Replikate ist zu jedem Zeitpunkt identisch. Um bestimmen zu können, welches die letzte Schreiboperation ist, sind vollständig synchronisierte Uhren notwendig. Aufgrund der physikalischen Abweichung der jeweiligen Uhren ist strikte Konsistenz daher in einem verteilten System nicht möglich.

**Sequenzielle Konsistenz:** L.Lamport definierte 1979 [32] das Model der Se-

quenziellen Konsistenz. Dabei muss das Ergebnis eines parallelen Ausführens aller Operationen identisch mit dem Ergebnis einer beliebigen sequeziellen Ausführung der Operationen sein. D.h. die tatsächliche Reihenfolge der Operationen ist nicht relevant, es muss lediglich gewährleistet werden, dass keine Wechselwirkung von Operationen vorkommen. Eine Möglichkeit, strikte Konsistenz zu erhalten, ist es, bei Schreibzugriffen *mutual exclusion* (mutex) zu verwenden [1] . Dabei blockiert jeder schreibende Knoten die entsprechende Variable bzw. Datei, bis der Schreibvorgang abgeschlossen ist. Dieses Blockieren erfordert jedoch das Versenden von mindestens zwei Nachrichten. Eine Nachricht zum Sperren, eine zum Entsperren des Objektes. Beide Nachrichten müssen jeweils bestätigt werden. Für Echtzeitoperationen wie Bewegungen oder schnelle Interaktionen würde das Einhalten von sequenzieller Konsistenz eine starke Verzögerung bedeuten. Ein kontinuierlicher Spielfluss wäre dabei nicht möglich.

**Eventual Consistency:** Bei diesem Modell wird der Konsistenzbegriff abgeschwächt, indem er von einem konkreten Zeitpunkt entkoppelt wird. Es wird lediglich gefordert, dass die Zustände der Objekte nach Beenden aller Schreiboperationen zu irgendeinem späteren Zeitpunkt identisch sein müssen.

In serverbasierten virtuellen Welten wird meist eine Kombination aus sequenzieller Konsistenz und eventual consistency verwendet. Für spielerelevante Operationen, die keine maximalen Echtzeitanforderungen haben (z.B. das Handeln mit Ware, das Verwalten des Spielerinventars, etc.) wird sequenzielle Konsistenz gefordert, bei schnelle Bewegungen und Interaktionen dagegen werden kurzzeitige Inkonsistenzen toleriert, und es wird lediglich eventual consistency gefordert. Diese striktes Aufteilung ist allerdings nicht immer möglich, da es durchaus spielerelevante Operationen mit hohen Echtzeitanforderungen geben kann. Ein Beispiel dafür sind Spiele wie Diablo 3 oder Path of Exile. Diese zeichnen sich durch eine sehr hohe Dynamik aus, bei der Spieler mehrere Aktionen pro Sekunde ausführen können. Zusätzlich gibt es einen Experten-

[1] Mutexe werden in der Regel auf Speicherebene verwendet. Das Konzept gilt jedoch ebenso für Objektzustände in verteilten Systemen

modus, bei dem ein Spieler nur ein Leben hat. Stirbt ein Spieler, muss er von Anfang an neu beginnen. Gute Spieler schaffen es dabei, hunderte Stunden Spielzeit zu überleben. D.h., jede Aktion, die zum Tod eines Spielers führt oder den Tod verhindern kann, ist spielerelevant. Dennoch wird aufgrund der hohen Echtzeitanforderungen nur eventual consistency verwendet. Das Ergebnis ist, dass bei hoher Serverlast oder einer schlechten Netzwerkverbindung Server und Client nicht synchronisiert sind. Dieser Zustand wird als *desync* bzw. als *out of sync* bezeichnet. Desync ist eine häufige Ursache für das Sterben von Spielern, was besonders unangenehm für Spieler im Expertenmodus ist.

In verteilten virtuellen Welten besteht für die Konsistenz der Zustände eine zusätzliche Herausforderung, da es keine zentrale Instanz gibt, die die Wechselwirkungen der Operationen unterbindet. So kann eine Server immer eine sequenzielle Ordnung der Operationen vorgeben. Auch bei einem eventual consistency Modell kann der Server den entsprechenden Zustand einfach vorgeben. In einer serverlosen virtuellen Welt müssen sich die beteiligten Knoten auf eine Reihenfolge der Operationen (sequenzielle Konsistenz), bzw. einen Zustand (eventual consistency) einigen.

### Korrektheit

Korrektheit und Konsistenz sind eng miteinander verbunden. Dennoch drücken sie unterschiedliche Sachverhalte aus und erfordern unterschiedliche Mechanismen für ihre Umsetzung. Im Gegensatz zur Konsistenz, bei der das relative Verhältnis von Zuständen relevant ist, muss für die Korrektheit der absolute Wert eines Zustandes interpretiert werden. Den Unterschied kann man sich leicht an einem Beispiel klarmachen:

Angenommen, zwei Spieler finden in einer virtuellen Welt eine Schatztruhe. Beide Spieler wollen die Truhe öffnen und den Schatz für sich beanspruchen. Je nach Definition gibt es verschiedene Möglichkeiten eines konsistenten Ausgangs der Situation. Spieler A oder Spieler B können beispielsweise den Inhalt erhalten. Ebenso können beide oder keiner die Beute bekommen. Alle diese Zustände wären konsistent, solange sie bei beiden Spielern identisch sind. Korrekt hingegen wäre nur der Zustand, in dem derjenigen Spieler die Beute bekommt, der zuerst auf die Kiste zugegriffen hat.

In diesem Beispiel sind Konsistenz und Korrektheit triviale Eigenschaften. In einer realen Spielsituation sind jedoch deutlich schwierigere Bedingungen gegeben, da einerseits keine absolute Synchronisation der jeweiligen Uhren möglich ist und andererseits sich in einer serverlosen Welt alle beteiligten Knoten über den Ausgang der Situation einigen müssen. D.h., alle beteiligten Knoten müssen eine Konsensentscheidung treffen. Eine Konsensbildung in Echtzeit von mehreren unsychronisierten Knoten ist jedoch ein äußerst anspruchsvolles Problem.

Um zumindest für theoretische Analysen eine formale Definition nutzen zu können, haben Mauve et al. [38] eine virtuelle, perfekte Maschine eingeführt. Diese Maschine hat vollständiges Weltwissen, und sämtliche Updatenachrichten werden ohne Verzögerung erhalten. Somit ist diese Maschine immer in der Lage, einen korrekten Weltzustand zu erzeugen. Die Korrektheit eines Systems bzw. einer verteilten virtuellen Welt wird dann als Konsistenz mit der perfekten virtuellen Maschine definiert.

### Robustheit

Allgemein beschreibt Robustheit, die Fähigkeit eines Systems, mit Fehlern umzugehen. Für verteilte virtuelle Welten bedeutet das in erster Linie die Fähigkeit, eine zusammenhängende konsistente virtuelle Welt zu verwalten, selbst wenn einzelne Knoten während des Betriebs ausfallen. Zusammenhängend bedeutet hier, dass alle Knoten durch das Overlaynetzwerk miteinander verbunden sind und keine Partitionierungen auftreten.

### Persistenz

Unter Persistenz versteht man die Dauerhaftigkeit eines Zustandes. Das bedeutet, dass ein Objekt nicht von einem dedizierten Knoten oder Server abhängen darf, der ja ausfallen kann. D.h., dass der Zustand des Objektes fortlaufend verwaltet wird. Fällt der verantwortliche Knoten aus, muss ein anderer Knoten für ihn einspringen. Um Persistenz zu ermöglichen, müssen immer mehrere Replikate der Objekte verwaltet werden.

### Responsiveness

Responsiveness bezeichnet die Fähigkeit eines Systems, auf eine Eingabe zu reagieren. Das kann die Antwort auf eine Suchanfrage oder das Lösen einer Gleichung sein. In Computerspielen ist damit konkret die Zeit gemeint, die benötigt wird, um eine Aktion eines Spielers zu erkennen und in der virtuellen Welt darzustellen. Diese Zeit wird maßgeblich von der Verzögerung des Netzwerkes beeinflusst, da Ein- und Ausgabegeräte und die Berechnung des Weltzustandes in der Regel nur Mikrosekunden benötigen. Wie schnell ein Netzwerk reagieren muss, um eine flüssiges Spielgefühl zu erzeugen, hängt dabei von mehreren Faktoren ab. Claypool et al. [12] haben dafür Aktionen in einer virtuellen Welt anhand der *Präzision*, mit der sie ausgeführt werden, und der *Deadline*, die bestimmt, wann eine Aktion abgeschlossen sein muss, eingeteilt. Sie kamen zu folgendem Ergebnis: Je höher die notwendige Präzision einer Aktion (z.B. Werfen einer Granate vs. Schießen mit einem Scharfschützengewehr) und je kürzer die Deadline (z.B. Bauen eines Hauses in einer Simulation vs. Schießen mit einem Gewehr) ist, desto schneller muss das Netzwerk reagieren. Henderson et al. [22] haben in ausführlichen Studien, in denen sie monatelang das Spielerverhalten von Half-Life-Spielern untersuchten, herausgefunden, dass neben der Spielsituation auch die Latenz anderer Spieler einen Einfluss auf das eigene Spielvergnügen hat. So ergab sich in Testreihen, in denen allen Spielern gleichmäßig Latenzen hinzugefügt wurden, eine geringere Austrittsrate aus dem Spiel als in Testreihen, in denen nur einzelnen Spielern Latenzen hinzugefügt wurden. Allgemein stellen Präzision, Deadline und relative Latenz die wichtigsten Einflussfaktoren dar, die die Bedeutung der Netzwerkverzögerungen bestimmen.

### Skalierbarkeit

Skalierbarkeit bezeichnet allgemein die Fähigkeit eines Systems, mit wachsenden Mengen umzugehen. Das kann zum Beispiel eine steigende Anzahl der Nutzer, eine zunehmende Menge an Suchanfragen oder eine geografische Ausdehnung der Spielwelt sein. Dabei sind der Skalierbarkeit eines Systems immer dann Grenzen gesetzt, wenn Dienste oder Daten zentralisiert verwaltet werden. D.h., kein System, das einen zentralen Server verwendet, ist unbegrenzt

skalierbar. Unbegrenzte Skalierbarkeit kann dagegen durch eine verteilte Systemarchitektur erreicht werden. Dabei unterscheiden sich verteilte von zentralisierten Systemen durch folgende Eigenschaften [55]:

- Kein Knoten kennt den vollständigen Systemzustand.
- Entscheidungen werden basierend auf lokalem Wissen getroffen.
- Fällt ein Knoten aus, kann das System dennoch weiter existieren.
- Es gibt keine exakt synchronisierten Uhren.

Die ersten drei Punkte sind notwendige Bedingungen für die Skalierbarkeit, der vierte Punkt folgt aus der verteilten Struktur und muss daher von jedem verteilten System beachtet werden.

#### Fairness

Fairness bewertet die Chancengleichheit der einzelnen Spieler. Dies ist besonders in verteilten Systemen relevant, da durch ein heterogenes Leistungs- und Belastungsprofil ein unterschiedliches Spielverhalten bei den einzelnen Knoten entstehen kann. So können Knoten, die lediglich über eine schwache Netzwerkverbindung verfügen und zusätzlich noch für die Persistenz mehrerer Objekte verantwortlich sind, durch die Überlastung der Netzwerkverbindung Updateinformationen nur verzögert verarbeiten. Das führt dazu, dass das Spielgeschehen nur verzögert dargestellt werden kann und der Spieler an diesem Knoten im Spiel benachteiligt ist. Daher gilt es für ein Overlay, sowohl das Leistungs- als auch das Belastungsprofil zu beobachten und bei Bedarf Dienste zu migrieren, um freie Ressourcen nutzen zu können.

#### Sicherheit

Aufgrund der besonderen Netzwerkarchitektur spielt Sicherheit in verteilten virtuellen Welten eine besondere Rolle, da Knoten direkt miteinander kommunizieren können. Das heißt insbesondere, dass die Netzwerkadresse eines Gegenspielers bekannt sein muss. Durch diese besondere Situation ist es stets möglich, dass ein Gegner unabhängig von der aktuellen Spielsituation durch

eine externe Hackerattacke ausgeschaltet werden kann. Die einzige Möglichkeit, ein derartiges Vorgehen zu verhindern, besteht im Verbergen der direkten Kommunikationspartner durch die Verwendung von Zwischenknoten.

Neben der direkten Angreifbarkeit der Knoten besteht ein zweites Risiko, da die Welt nicht von einer neutralen Instanz (dem Server) verwaltet wird, sondern von den Spielern selbst: Nimmt ein korrupter Knoten an dem Spiel teil, ist er in der Lage, die gesamte Welt nach seinen Regeln zu verändern („cheating“).

Diese beiden Probleme sind fundamental. Sie sind eine der Ursachen dafür, dass es derzeit noch keine kommerzielle Version einer verteilten virtuellen Welt ohne Server gibt.

## 2.4 Netzwerkarchitektur

Die im vorangegangenen Abschnitt vorgestellten Qualitätskriterien definieren, welche Eigenschaften für ein MMOG-Netzwerk von Bedeutung sind. Um nun einen konstruktiven Ansatz zu entwickeln, werden unterschiedliche Teilelemente benötigt, die zu einer Netzwerkarchitektur zusammengefügt werden können. Eine derartige Architektur besteht aus einer Topologie, die von verschiedenen Mechanismen, Diensten und Protokollen genutzt wird. Eine strikte Trennung der einzelnen Komponenten ist dabei nicht immer möglich.

### 2.4.1 Netzwerktopologien

Unter einer Netzwerktopologie versteht man die Struktur der Verbindungen in einem Netzwerk. Dabei muss man zwischen realen, logischen und virtuellen Topologien unterscheiden. Eine reale Topologie wird durch die Form der physikalischen Netzwerkverbindungen definiert. So können beispielsweise Knoten in einem lokalen Netzwerk ringförmig miteinander verbunden werden. Jeder Knoten kann dann nur mit seinen direkten Nachbarn kommunizieren. Möchte ein Knoten mit weiter entfernten Knoten kommunizieren, müssen die Nachrichten von Knoten zu Knoten weitergereicht werden. Ebenso sind sternförmige, vermaschte, linienförmige, busförmige oder baumförmige Topologien möglich (siehe Abbildung 2.5).

Logische Topologien beschreiben den Fluss der Daten. Dieser kann von der Struktur der Verbindungen abweichen. So liegt beispielsweise bei der Verwendung eines Hubs in einem lokalen Ethernet-Netzwerk eine sternförmige physikalische, aber eine busförmige logische Verbindung vor. Die reale und die logische Netzwerktopologie sind bei virtuellen Online-Welten durch das Internet fest vorgegeben. Sie bestehen aus einer komplexen, dynamischen Kombination aus mehreren Teiltopologien.

Um dedizierte Kommunikationsgruppen bilden zu können, ist eine zusätzliche virtuelle Struktur notwendig. Da virtuelle Topologien über dem vorhandenen physikalischen, bzw. logischen Netzwerk gebildet werden, werden sie als Overlay bezeichnet. Ein Overlaynetzwerk kann ebenfalls sämtliche Formen annehmen. Die einfachste Form ist ein Client-/Server-Overlay. Es basiert auf einer Stern-Topologie, bei der alle Knoten (Clients) eine Verbindung zu einem zentralen Knoten (Server) aufbauen. Dieser kennt die Knoten des Netzwerkes und ist in der Lage, Nachrichten an bestimmte Knoten weiterzuleiten. Die Vorteile dieser Architektur bestehen darin, dass sie leicht zu implementieren ist und dass der Aufbau des Netzes und der Fluss der Daten von einer zentralen Stelle aus kontrolliert werden können. Nachteilig ist, dass die Sicherheit des Netzes von einem einzigen Knoten abhängt (single point of failure). Zusätzlich entsteht bei dem Server ein hohes Aufkommen an Netzwerkverkehr. Das ist zum einen mit Kosten verbunden, zum anderen kann so ein Flaschenhals im Netzwerk entstehen. Für Netzwerke kleiner bis mittlerer Größe sind Client-Server-Architekturen dennoch meist die geeignete Wahl.

Im Gegensatz zur Sterntopologie der Client-Server-Architektur sind alle weiteren Topologien nicht zentralisiert. D.h., es gibt keinen dedizierten Knoten, der den Aufbau des Netzes und den Fluss der Daten steuert. Beides muss von den Knoten selbst organisiert werden. Da sie dabei alle gleichberechtigt sind, spricht man von Peer-to-Peer-Netzwerken (P2P). Ein typisches Beispiel für ein P2P-Netzwerk ist *„Chord“* [53]. Hier werden die Knoten in einer virtuellen Ringtopologie angeordnet. P2P-Netzwerke werden meist für hoch-skalierbare Anwendungen verwendet. Weitere P2P-Architekturen werden im Abschnitt 2.5 vorgestellt.

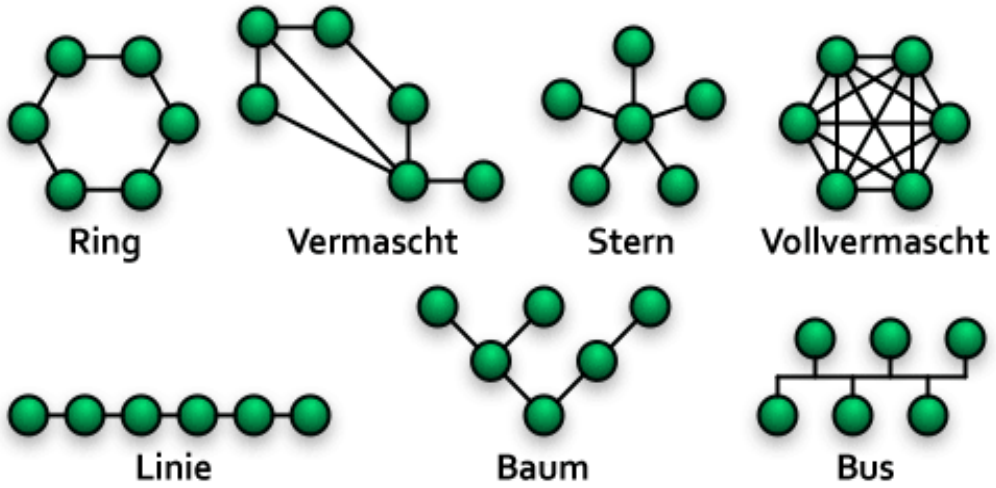



Abbildung 2.5: Überblick über Netzwerktopologien. (Bildquelle: www.wikipedia.de)

### 2.4.2 Netzwerkdienste

Netzwerkdienste sind Funktionen, die inhaltliche Aufgaben umsetzen. Ein typisches Beispiel dafür ist das Aktualisieren von Objektzuständen. Für verteilte Anwendungen gilt, dass sie deutlich mehr Netzwerkdienste benötigen, da sämtliche Steuerfunktionen der virtuellen Welt in der Lage sein müssen, zwischen Knoten zu migrieren. So erfordert beispielsweise die Implementierung einer *Künstlichen Intelligenz (KI)* auf einem Server keinen Netzwerkdienst. In einer P2P-Anwendung kann es hingegen vorkommen, dass der Knoten, der gerade einen KI-Spieler steuert, das Netz verlässt. Der Zustand und die Funktionalität des KI-Spielers müssen dann auf einen anderen Knoten übertragen werden. Fan et al. [18] haben die wichtigsten Netzwerkdienste einer verteilten virtuellen Welt identifiziert:

**Interest Management (IM):** IM ist der Dienst, der entscheidet, mit welchen Spielern kommuniziert werden muss, um die persönliche Sicht der Welt zu erzeugen. Diese Sicht wird im Spiel durch den Wahrnehmungs- und den Wirkungsbereich eines Spielers definiert. Der Wahrnehmungsbereich kann beispielsweise aus dem Sicht- und dem Hörbereich bestehen, der Wirkungsbereich aus dem Wirkungsbereich der spielespezifischen Waffen oder Aktionen. Da jeder dieser Bereiche eine komplexe und

dynamische Struktur haben kann, wird meist das Konzept einer *Area of Interests (AOI)* verwendet. Dieser in der Regel kreisförmig den Spieler umgebende Bereich definiert dann, welche Ereignisse von Interesse sind bzw. mit welchen Spielern Nachrichten ausgetauscht werden müssen. In P2P-Implementierungen werden hierfür meist direkte Verbindungen aufgebaut. Spieler innerhalb des AOI werden als direkte Nachbarn bezeichnet. Das Verwalten einer akkuraten und aktuellen Liste der Nachbarn ist die Hauptaufgabe eines verteilten IM.

**Game Event Dissemination (GED):** GED ist ein Dienst, der sicherstellt, dass jeder Spieler über alle für ihn innerhalb seiner AOI relevanten Ereignisse informiert wird. Der GED-Dienst ist dabei stark vom Genre eines Spiels abhängig. So erfordern Shooter in erster Linie geringe Verzögerungen bei der Verbreitung der Ereignisse, um so den Echtzeitcharakter und die Dynamik des Spiels zu erhalten. Die Implementierungen eines GED-Dienstes unterscheiden sich bei Client-/Server- und P2P-Architekturen dahingehend, dass Aggregation und Filterung von Daten bei einer Client-/Serverarchitektur zentral durchgeführt werden können. In P2P-Systemen muss dies bei den Peers geschehen.

**NPC Host Allocation:** Die meisten Spiele verwenden so genannte *Non-Player-Charaktere (NPC)*. Das sind Spieler, die von Skripten oder einer Künstlichen Intelligenz gesteuert werden und die in der Lage sind, steuernde Funktionen in der virtuellen Welt umzusetzen (z.B. Wachen, die verhindern, dass Spieler eine fremde Stadt betreten, oder Händler, die die Aufgabe haben, Märkte zu steuern). In verteilten Welten müssen diese NPCs auf den Knoten der Spieler ausgeführt werden. Die Aufgaben, Verantwortlichkeit und Zustände der NPCs müssen in der Lage sein, zwischen Peers zu migrieren.

**Game State Persistence:** Auch Objekte, die nicht direkt mit einem Spieler verbunden sind, müssen konsistent und persistent gespeichert werden. So muss beispielsweise der Inhalt einer Schatztruhe zu jeder Zeit für jeden Spieler identisch sein, selbst wenn der aktuell verantwortliche Knoten abrupt ausfällt. Dies lässt sich nur durch Replikate realisieren. Dabei ist es von zentraler Bedeutung, dass die Operationen auf diesen Replikaten mit

einer möglichst geringen Verzögerung durchgeführt werden, ohne dass sie direkt zu Verzögerungen des Spielflusses führen.

**Cheating Mitigation:** Da in verteilten virtuellen Welten der Zustand der Welt von den Spielern selbst verwaltet wird, ist ein spezieller Dienst erforderlich, der dafür sorgt, dass sich kein Spieler betrügerisch verhalten kann. So können Spieler beispielsweise mit einer *Denial of Service (DoS)-Attacke* die Netzwerkverbindung eines anderen Spielers blockieren und sich somit einen Vorteil verschaffen. Da kein Knoten eine globale Sicht hat, muss auch dieser Dienst deshalb über das Netzwerk synchronisiert operieren.

**Incentive Mechanisms:** P2P-Systeme nutzen die Ressourcen der Teilnehmer für den Aufbau des Netzwerks. Das beinhaltet Netzwerkbandbreite, CPU-Leistung und Speicher. Die Menge der nutzbaren Ressourcen beeinflusst dabei maßgeblich die Qualität des Netzwerkes. Jedoch kann nicht davon ausgegangen werden, dass jeder Knoten freiwillig seine Ressourcen für andere freigibt. Daher ist ein Dienst notwendig, der Anreize für Knoten schafft, Ressourcen für das Netzwerk bereitzustellen.

Für das Betreiben einer verteilten virtuellen Welt sind IM und GED die beiden grundlegenden Dienste.

### 2.4.3 Netzwerkmechanismen

Im Gegensatz zu Netzwerkdiensten sind Netzwerkmechanismen Funktionen, die Lösungen für technische Anforderungen umsetzen. So ist es beispielsweise nicht möglich, dass in einem Netzwerk alle Knoten den identischen Weltzustand haben, da Updatenachrichten stets eine physikalische Distanz überwinden müssen und dafür Zeit benötigen. Um derartige Verzögerungen so gering wie möglich zu halten bzw. ihre Auswirkungen zu kaschieren, können unterschiedliche Mechanismen eingesetzt werden. Ebenso gibt es Mechanismen für die Reparatur fehlerhafter Zustände oder Verfahren, die ein unfaires Ausnutzen von Verzögerungen verhindern.

**Dead Reckoning - Guided AI:** Aufgrund der Tatsache, dass das Internet keine gesicherte Dienstqualität bereitstellen kann, kommen Updatenach-

richten nie mit einer konstanten Taktfrequenz an. Das bedeutet, dass beispielsweise Informationen über Bewegungen von Mitspielern mit einer gewissen Varianz empfangen werden. Würde immer direkt die neu erhaltene Position eines Spielers anzeigt, hätte das ruckartige Bewegungen zur Folge, insbesondere, wenn Pakete stark verzögert ankommen. Daher wird für viele kontinuierliche Aktionen eine Technik verwendet, die basierend auf dem aktuellen Wissen einen zukünftigen Zustand extrapoliert. Läuft beispielsweise ein Spieler in eine bestimmte Richtung, wird solange eine Bewegung in diese Richtung angezeigt, bis ein neues Positionsupdate eintrifft. Trifft eine neue Updatenachricht dann ein, wird der Weg zu der neuen Position interpoliert und in den nächsten Frames angezeigt. Die Bezeichnung für diese Technik basiert auf einem historischen Prinzip der Seefahrt, das zur Navigation genutzt wurde, wenn die aktuelle Position nicht anhand der Sterne bestimmt werden konnte. In Anlehnung an dieses Prinzip wird sie daher *Dead Reckoning* genannt. Diese meist lineare Extrapolation kann auch durch die Verwendung einer KI verbessert werden, die in der Lage ist, komplexe Bewegungsmuster von Spielern zu lernen *(Guided AI)* [7].

Da die Verzögerungen im Allgemeinen deutlich unter einer Sekunde betragen und Spieler die tatsächliche Position der Mitspieler gar nicht kennen, entsteht durch diese Techniken eine glatte Bewegung, die selbst bei schlechten Netzwerkbedingungen immer noch ein hinreichend präzises Bild der tatsächlichen Spielsituation wiedergibt.

**Local Lag - Time Warp:** Durch die Netzwerkverzögerungen können inkonsistente Zustände bei einzelnen Knoten entstehen. Dabei gibt es unterschiedliche Ursachen, die unter Umständen zu schwerwiegenden Fehlern führen können. Um diese Fehler vermeiden oder beheben zu können, ist es zunächst notwendig, die unterschiedlichen Formen der möglichen Fehler zu verstehen. Dies kann an einem einfachen Beispiel demonstriert werden, das bereits von Mauve et al. [38] verwendet wurde:

Angenommen, zwei Spieler (A und B) spielen gemeinsam eine Eisenbahnsimulation. Ein Zug nähert sich einer Weiche, die Spieler B kontrolliert und die zwischen Gleis 1 und Gleis 2 umschalten kann. Ursprünglich steht

die Weiche auf Gleis 1. Im einfachsten Fall stellt Spieler B die Weiche auf Gleis 2. Die Nachricht benötigt $100ms$, um Spieler A zu erreichen. In diesen $100ms$ nähert sich der Zug der Weiche, überfährt sie jedoch nicht. Spieler A kann in seiner Welt rechtzeitig die Umstellung der Weiche anzeigen, und der Zug auf Gleis 2 weiterfahren. Beide Spieler haben anschließend wieder einen konsistenten Zustand. Diese Art von Inkonsistenz ist lediglich durch das Netzwerk bedingt und hat keine weiteren Folgen für den Spielfluss. Da keine weiteren Maßnahmen notwendig sind, um die Inkonsistenz zu beheben, wurde sie von Mauve et al. nicht als Inkonsistenz betrachtet.

Kritischer wird die Situation, wenn der Zug bei Spieler A die Weiche bereits überfahren hat, wenn die Nachricht von Spieler B eintrifft. Denn bei Spieler A ist der Zug auf Gleis 1 und bei Spieler B auf Gleis 2 weitergefahren. In diesem Fall muss der lokale Zustand repariert werden. Das bedeutet, dass ein Sprung zwischen den unterschiedlichen Zeitlinien durchgeführt werden muss. Dazu muss Spieler A beim Erhalten der Nachricht von B nachträglich seinen Zustand korrigieren. Der neue Weltzustand wird dann in einem *Fast Forward Modus* direkt berechnet und angezeigt. Für das Beispiel bedeutet das, dass der Zug, der zunächst bei Spieler A noch 100 Meter auf Gleis 1 gefahren ist, einen Sprung macht und sich 100 Meter weiter auf Gleis 2 befindet. Diese Art der Reparatur wird *Time Warp* genannt. Sie hat zwar einen harten Eingriff in das Spielgeschehen zur Folge (springender Zug), aber der zeitliche Fluss der Aktionen bleibt erhalten, und Eventual Consistency kann erreicht werden.

Für Spieler A stellt der Time Warp jedoch einen starken Eingriff in den Spielfluss dar. Daher gilt es, Time Warps nach Möglichkeit zu vermeiden. Eine Technik, die dafür eingesetzt werden kann, ist das lokale Verzögern von relevanten Ereignissen (*Local Lag*). Dabei werden Ereignisse, deren Deadline es zulässt, lokal nicht direkt angezeigt, sondern etwa um die Dauer der Netzwerkverzögerung zurückgehalten. Im Falle der Weiche würde der Zug bei beiden Spielern ca. $100ms$ verzögert angezeigt und dadurch die kurzzeitige Inkonsitenz vermieden werden. Allgemein kann so durch die Verwendung von Local Lag die Anzahl der benötigten Time

Warps reduziert werden. Jedoch gilt es dabei immer einen geeigneten Trade-off zwischen Konsistenz und Responsiveness zu finden.

**Lockstep:** Aufgrund der Tatsache, dass in verteilten virtuellen Welten die Knoten selbst für den Weltzustand verantwortlich sind, kann es zu unfairen Eingriffen in den Ablauf des Spiels von Seiten einzelner Knoten kommen. Um kritische Entscheidungen treffen zu können, wird dabei in der Regel eine Abstimmung aller beteiligten Knoten genutzt. Dazu werden oft einfache *Stop-and-Wait-Mechanismen* verwendet.

In einem Beispielszenario befinden sich mehrere Spieler in einem Gefecht. Ausgangspunkt ist der Zeitpunkt $t_0$. Nun führen alle Knoten ihre nächste Aktion aus und schicken die entsprechenden Updatenachrichten an alle beteiligten Knoten. Die Knoten warten so lange, bis sie alle aktuellen Updatenachrichten empfangen haben. Gemeinsam einigen sie sich über den neuen Weltzustand zum Zeitpunkt $t_1$ und zeigen diesen jeweils lokal an.

Dieses Vorgehen sorgt zwar dafür, dass sich die Knoten auf einen zeitlichen Ablauf der Ereignisse einigen, jedoch kann es ebenso für unfaire Spielzüge missbraucht werden. So kann ein unfairer Knoten eine Funktion implementieren, die auf die Nachrichten aller anderen Knoten wartet und eine entsprechende Reaktion ausführt. Zum Beispiel kann der Knoten jeden Angriff mit einem Ausweichmanöver seines Avatars beantworten, das zur gleichen Zeit stattgefunden hat. Um diese *Lookahead-Attacken* zu verhindern, haben Baughman et al. [6] das *Lockstep-Verfahren* entwickelt. Dabei verschicken die Knoten zunächst einen Hash-Wert, der aus der aktuellen Aktion gebildet wird. Nachdem alle Hashes erhalten wurden, werden die Updatenachrichten als Klartext versendet. Durch dieses zweistufige Verfahren wird gewährleistet, dass alle Knoten ihre Aktionen nur basierend auf dem Weltzustand $t_0$ ausführen. Allerdings gilt auch hier, dass das Verfahren ein Trade-off zwischen Sicherheit und Responsiveness ist, da dabei zwei Updatenachrichten aller beteiligten Knoten verschickt werden müssen, bis eine Aktion ausgeführt werden kann.

### 2.4.4 Netzwerkprotokolle

Neben der Verwendung unterschiedlicher Topologien, Dienste und Mechanismen stellt sich noch die Frage, welches Transportprotokoll für die Übertragung über das Internet genutzt werden soll. Dabei ist die entscheidende Frage, ob ein sicheres, strombasiertes Protokoll (TCP) oder ein unsicheres, datagrammbasiertes Protokoll (UDP) verwendet werden soll. Allgemein lässt sich sagen, dass spiele-relevante Nachrichten immer ein sicheres Protokoll verwenden sollten. Denn wenn Pakete verloren gehen, führt das zu einer Verzerrung der Spielrealität. Zeitkritische Pakete sollten dagegen immer als Datagramm verschickt werden. Dabei hängt es oft von der Situation oder dem Spiel ab, welche Nachrichten relevant sind und welche nicht, und wie echtzeitkritisch die Informationen sind. So ist bei Shootern beispielsweise die Information eines Schusses sehr echtzeitkritisch, da Menschen bei der Wahrnehmung von kleinen schnellen Objekten wie Projektilen sehr sensibel gegenüber Verzögerungen sind. Daher wird meist UDP verwendet, obwohl die Information eines Schusses spiele-relevant ist. Ein weiterer Grund dafür ist, dass in Shootern oft eine hohe Anzahl von Kugeln verschossen wird und so der Verlust eines einzelnen Updatepaketes meist nicht auffällt. Im Gegensatz dazu sind bei MMOGs einzelne Schüsse bzw. Anwendungen von Fähigkeiten wie Zaubersprüchen von hoher Bedeutung. So kann beispielsweise der Verlust eines einzelnen Heilspruchs den Ausgang eines 30-minütigen Kampfes beeinflussen, an dem 40 Spieler beteiligt sind. Zusätzlich sind die Fähigkeiten meist mit aufwändigen grafischen Effekten verbunden, deren Ausbleiben jedem Spieler sofort auffallen würde. Daher wird in den meisten MMOGs ausschließlich TCP verwendet. Allgemein gilt, dass die Entscheidung über das Transportprotokoll und die dabei verwendeten Protokollparameter stark von der Art einer Updatenachricht abhängen. Spielentwickler betreiben oft einen großen Aufwand, um die Protokolle bestmöglich an die Anforderungen der Nachrichten anzupassen.

## 2.5 Implementierungen

Eine vollständige Umsetzung einer unbegrenzt skalierbaren virtuellen Welt existiert derzeit nicht. Dennoch gibt es eine Vielzahl kommerzieller Implementierungen und Forschungsprojekte, die sich als Ziel gesetzt haben, eine

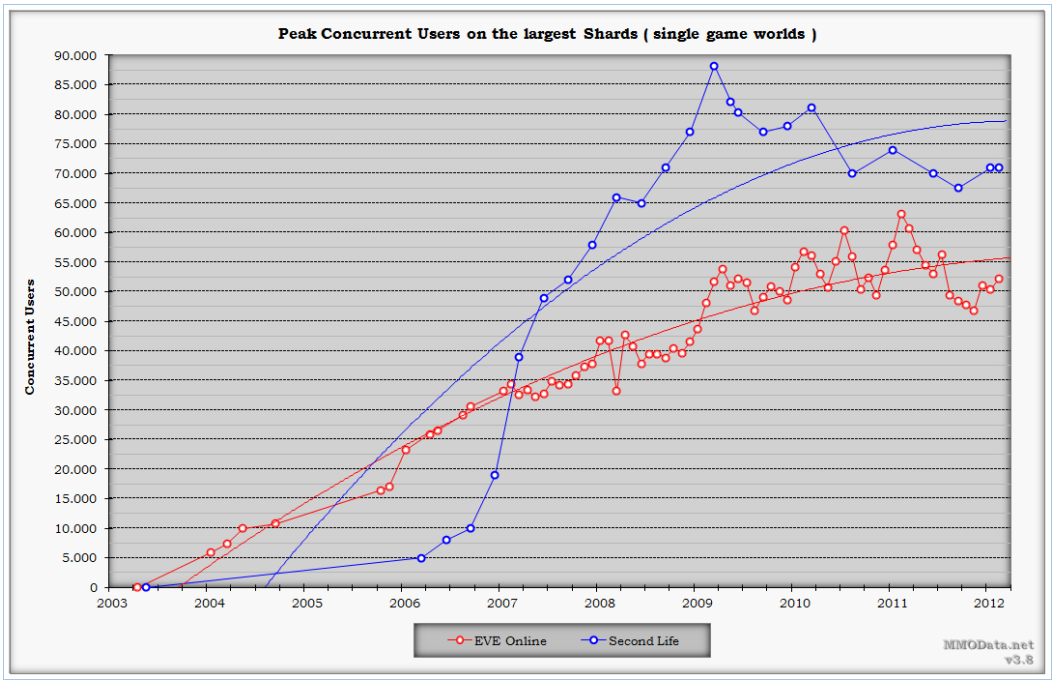



Abbildung 2.6: Übersicht über zeitgleiche Nutzer von Eve Online und Second Life, die sich in der selben Spielwelt (shard) befinden. Der aktuelle Rückgang der Nutzerzahlen ist mit der Verwendung von neuer Software zur Erkennung automatischer Spieler und Real Money Trader (RMT) zu erklären. (Quelle:`http://users.telenet.be/mmodata/Charts/PCUShard.png`)

virtuelle Welt für möglichst viele Teilnehmer zu schaffen. Dabei verfolgen kommerzielle Anwendungen den Ansatz der Hardwareskalierung. D.h., es werden extrem leistungsfähige Server und Netzwerkverbindungen eingesetzt, um die maximal mögliche Teilnehmerzahl zu erhöhen. Aus gewerblicher Sicht ist diese Vorgehensweise sinnvoll, da einerseits nur mit einer Serverarchitektur die volle Kontrolle der Anwendung in den Händen der Entwickler liegt, andererseits der rasante Fortschritt der Hardware ein kontinuierliches Wachsen der Welten ermöglicht. Virtuelle Welten wie *Eve Online* oder *Second Life* haben es mit diesem Ansatz bereits geschafft, gleichzeitig über 50.000 Spieler in einer einzigen Welt zu verwalten (siehe Abbildung 2.6). Dennoch zeigt sich deutlich die Grenze des Machbaren, da es insbesondere bei großen Schlachten stets zu starken Verzögerungen kommt, die ein normales Spielen kaum noch möglich machen. Darüber hinaus entstehen für die Anschaffung der Hardware und den monatlichen Betrieb Kosten in Höhe von vielen Millionen US Dollar.

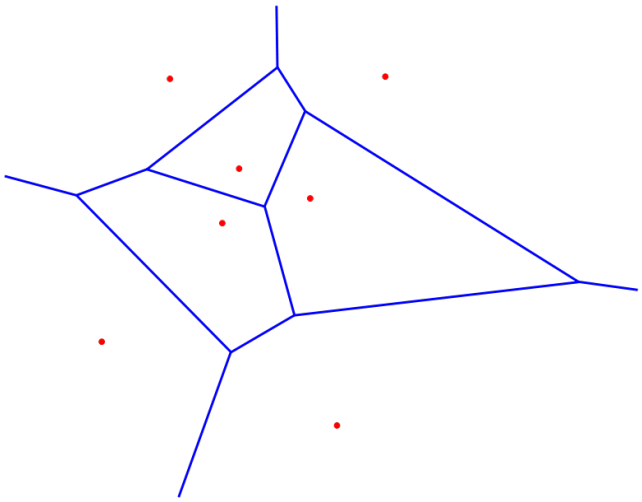


Abbildung 2.7: Voronoidiagramm einer Zone mit sieben Spielern. Jedem Spieler wird die Region zugeordnet, die sich näher zu ihm als zu den anderen Spielern befindet. (Bildquelle:`http://de.wikipedia.org/wiki/Voronoi-Diagramm`)

Im Gegensatz zu den kommerziellen Anwendungen verfolgen einige Forschungsprojekte den Ansatz, das Problem der Skalierbarkeit und der hohen Kosten grundsätzlich zu lösen, indem eine vollständig verteilte Netzwerkarchitektur verwendet wird. Das bedeutet jedoch, dass wie oben erwähnt, die gesamte Verwaltung der virtuellen Welt von den Teilnehmern selbst realisiert werden muss, was mit sehr hohem logistischem Aufwand verbunden ist. Dieser Aufwand lässt den Umfang eines derartigen Projektes im Vergleich zu einer einfachen Client-/Serverarchitektur um ein Vielfaches anwachsen. Dennoch gibt es bereits verschiedene prototypische Implementierungen, die den Einsatz eines Overlaynetzwerkes für ein MMOG erproben. Im Folgenden werden zwei prominente Ansätze explizit und weitere zusammenfassend vorgestellt.

### 2.5.1 Voronoi-based Overlay Network - VON

Als eines der ersten serverlosen Overlaynetzwerke für MMOGs wurde 2004 von Hu et al. [24, 23] das *Voronoi-based Overlay Network (VON)* vorgestellt. Dieser Ansatz basiert auf einer Aufteilung der Welt in $N$ Regionen, wobei $N$ gleich der Anzahl der Spieler ist.

Eine Region $R_n$ definiert sich durch die Menge alle Punkte, deren Abstand zu Spieler $S_n$ geringer ist als der zu allen anderen Spielern. So wird jedem

Spieler eindeutig diejenige Zone zugeordnet, zu der er sich am nächsten befindet (siehe Abbildung 2.7). Diese Aufteilung ist eindeutig und kann von jedem Spieler lokal berechnet werden. Lokal bedeutet dabei, dass nicht das Voronidiagramm für die gesamte Welt berechnet wird, sondern nur für den Ausschnitt der Welt, den ein Knoten „sehen“ kann. Die Sichtweite wird dabei durch die AOI festgelegt. Zu innerhalb der AOI liegenden und angrenzenden Knoten werden direkte Verbindungen aufgebaut und Updatenachrichten ausgetauscht. Angrenzende Nachbarn können dann in regelmäßigen Abständen nach Veränderungen und neuen Nachbarn befragt werden. Somit ist gewährleistet, dass bei Bewegung der Knoten Spieler von außerhalb erkannt werden und dass rechtzeitig eine Verbindung zu dem entsprechenden Knoten aufgebaut werden kann (siehe Abbildung 2.8).

Um einem VON beizutreten, muss ein beliebiger Knoten daraus bekannt sein. An diesen Knoten wird eine *Join-Anfrage* geschickt, die dann in die Richtung der Position des beitretenden Knotens weitergeleitet wird. Dazu wird ein geografisches Routing verwendet, dass auf dem Voronoidiagramm basiert. Erreicht die *Join-Anfrage* den Knoten, der für die Beitrittszone verantwortlich ist, kann dieser die Anfrage mit einer aktuellen Nachbarliste beantworten. Der beitretende Knoten kann dann die Nachbarn über seinen Beitritt informieren und den aktuellen Weltzustand erfragen (siehe Abbildung 2.9). Weitere Details über die Funktionsweise von VON können [24] entnommen werden.

### 2.5.2 pSense

Ebenso wie VON ist pSense ein Overlaynetzwerk, das für den Interest-Management- und Game-Event-Dissemination-Dienst eingesetzt werden kann. Es wurde am Lehrstuhl für *Databases and Distributed Systems (DVS)* der TU Darmstadt entwickelt. Im Gegensatz zu VON wird die Welt jedoch nicht durch ein Voronoidiagramm aufgeteilt, sondern durch die AOI und Zonen, die durch bestimmte Winkelabschnitte definiert werden. Dabei werden Knoten, die sich innerhalb der AOI befinden, als *Near Nodes* bezeichnet. Zu diesen *Near Nodes* wird eine direkte Verbindung aufgebaut, und Updatenachrichten werden ausgetauscht. Dabei können verschiedene Kommunikationsschemata verwendet werden, die die Updaterate dynamisch in Abhängigkeit zur Spielsituation und der zur Verfügung stehenden Bandbreite adaptieren. Ausgewählte Kno-

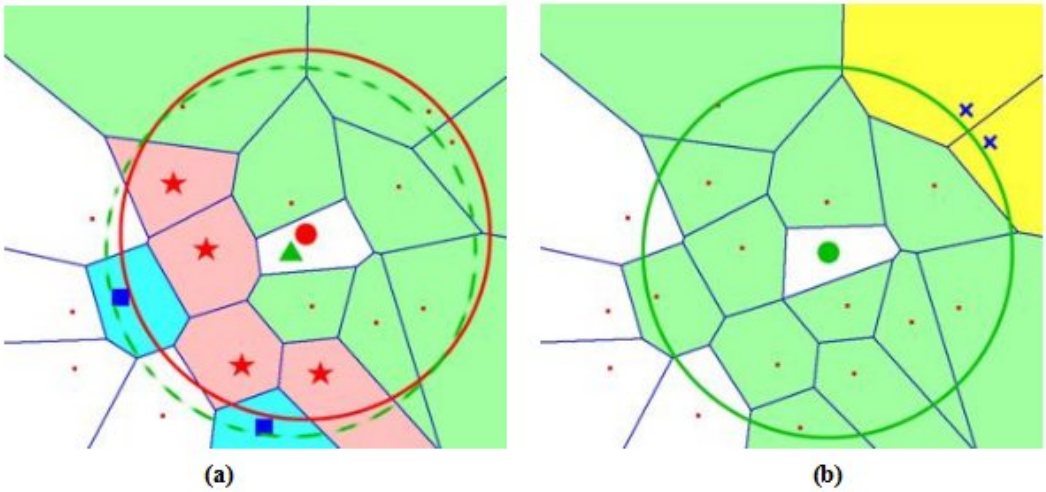



Abbildung 2.8: Update des VON bei sich bewegenden Knoten: (a) Das grüne Dreieck zeigt die neue Position des sich bewegenden Knotens. Mit blauen Quadraten markiert sind die neuen Nachbarn, die durch Befragung der Grenznachbarn (Sterne) erkannt werden müssen. (b) Nach der Bewegung können nicht mehr benötigte Verbindungen abgebaut werden (Kreuze).(Bildquelle:`http://vastlib.wikispaces.com/VON`)

ten, die sich außerhalb der AOI befinden, werden als *Sensor Nodes* festgelegt. Mit diesen *Sensor Nodes* werden Nachrichten in einer geringen Updatefrequenz ausgetauscht (ca. eine Nachricht pro Sekunde). Ziel der *Sensor Nodes* ist es, einen Knoten zu informieren, wenn sich andere Spieler seiner AOI nähern. Um sicherzustellen, dass alle neuen Nachbarn erkannt werden, müssen die Sensorknoten derart gewählt werden, dass sie gleichförmig in allen Richtungen verteilt sind. Dazu wird bei pSense die Umgebung außerhalb der AOI in konstante Winkelabschnitte unterteilt. Jeder Abschnitt enthält dabei einen eigenen *Sensor Node* (siehe Abbildung 2.10).

Der Beitritt zu einem pSense-Netzwerk erfolgt ebenso wie bei VON. Es wird zunächst ein bekannter Knoten kontaktiert. Dieser leitet die Anfrage über seine *Sensor Nodes* in die entsprechende Beitrittszone weiter. Dort werden dann die entsprechenden Knoten über den Beitritt informiert, und die *Near Nodes* sowie die *Sensor Nodes* des neuen Knotens müssen festgelegt werden. Weitere Details über die Funktionsweise von pSense können [49] entnommen werden.

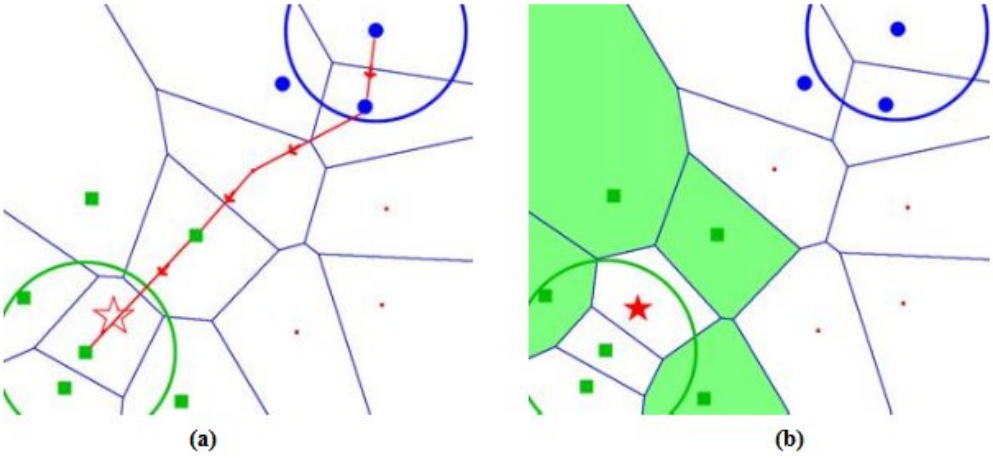



Abbildung 2.9: Beitritt eines Knotens: (a) Die blauen Punkte stellen den Zugangsknoten und seine Nachbarn dar. Der für die Beitrittszone verantwortliche Knoten und seine Nachbarn sind durch die grünen Quadrate symbolisiert. Die Beitrittsanfrage wird geografisch in die Zielzone geroutet (roter Pfad). (b) Der rote Stern zeigt die Position, an der der neue Knoten eingefügt wird. Alle direkten und angrenzenden Nachbarn müssen informiert werden.(Bildquelle:`http://vastlib.wikispaces.com/VON`)

### 2.5.3 Weitere Overlaystrukturen

Neben VON und pSense gibt es eine Vielzahl unterschiedlicher Ansätze, verteilte Overlaystrukturen als skalierbare Netzwerkarchitektur für Spiele einzusetzen. Dabei gibt es drei unterschiedliche Strategien, anhand derer die Knoten kommunizieren:

**Gegenseitige Benachrichtigung:** Knoten beobachten Veränderungen der virtuellen Welt nicht nur aus ihrer Sicht, sondern auch aus der ihrer direkten Nachbarn. So können sie erkennen, ob neue Spieler in den Sichtbereich eines Nachbarn gelangen, und diesen rechtzeitig informieren. Neben VON gibt es weitere Ansätze, die dabei auf eine Unterteilung der Welt durch ein Voronoidiagramm zurückgreifen, wie beispielsweise das Nomad Overlay von Ricci et al. [45] oder Solipsis von Keller et al. [30]. Sie unterscheiden sich hauptsächlich durch abgewandelte *Join-/Leave-Meachnismen* oder die Verwendung unterschiedlicher Transportprotokolle. Ebenso wie pSense unterteilt QuON [4] die Umgebung in Unterregionen. Jedoch werden bei QuON keine Winkelabschnitte, sondern Qua-

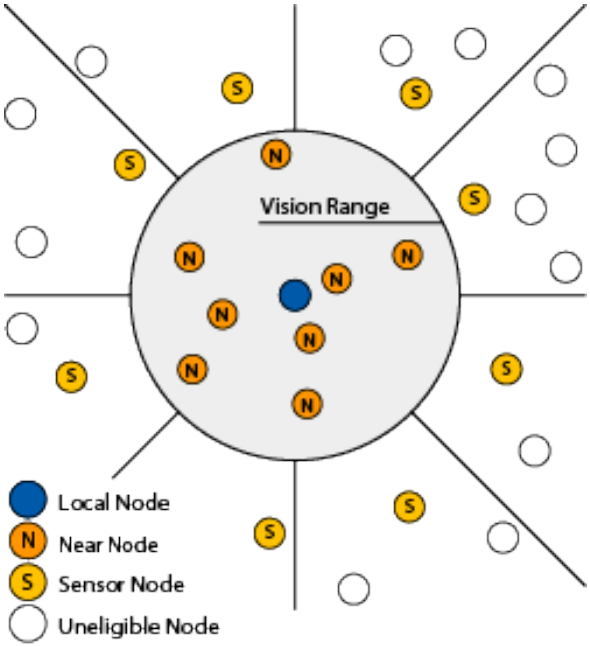



Abbildung 2.10: Das pSense-Overlay teilt die Knoten in drei Gruppen: Near Nodes, die sich innerhalb der AOI befinden, Sensor Nodes, von denen sich jeweils einer in einer festgelegten Richtung befindet, und Knoten, zu denen keine Verbindung aufgebaut wird (Uneligible Nodes). (Bildquelle:`http://www.dvs.tu-darmstadt.de/research/psense/`)

dranten in Verbindung mit einem Quad-Tree-Algorithmus verwendet.

**Austausch von Nachbarschaftslisten:** Im Gegensatz zur *Gegenseitigen Benachrichtigung* werden beim Austausch von Nachbarschaftslisten die Informationen nicht für die Nachbarknoten interpretiert und gefiltert, sondern vollständige Knotenlisten unter den Nachbarn ausgetauscht. Dieser Ansatz ist einfacher zu implementieren, erzeugt jedoch deutlich mehr Netzwerkverkehr. Bekannte Vertreter sind *P2P Message Exchange* von Kawahara et al. [29] und das *Message Interchange Protocol* von Chen et al. [11]

**Zonen-basierte Koordinatoren:** Die ersten beiden Strategien basieren auf einer dynamischen Kommunikationsstruktur, die kontinuierlich an die aktuelle Situation der Spielwelt angepasst wird. Die Erkennung der Nachbarn erfolgt ausschließlich durch die aktuelle Position der Spieler. Im Gegensatz dazu nutzen einige Verfahren zonenbasierte Koordinatoren.

Das sind Knoten, die für bestimmte Regionen die Aufgabe der Nachbarschaftserkennung übernehmen. Diese Superpeer-Rolle wird zwar in den meisten Ansätzen ebenso dynamisch zugewiesen, dennoch ergibt sich zeitweise eine statische (lokale) Client-/Server-Architektur. Dabei unterscheiden sich die einzelnen Ansätze in erster Linie in der Festlegung der Zonen [31, 27, 56, 19].

Leistungsmerkmale der einzelenen Verfahren lassen sich zunächst nur durch eine Analyse der Mechanik ableiten. So gilt beispielsweise für Zonen-basierte Koordinatoren, dass die Kommunikation innerhalb einer Zone mittels eine Client/-Server-Architektur umgesetzt wird. Das bedeutet, dass Konsistenzmodelle einfach umgesetzt werden können. Im Gegenzug dazu werden einzelne Knoten mit einer hohen Last belegt, was zu einer Benachteiligung dieser Knoten führen kann. Zusätzlich muss ein aufwändiger Mechanismus zur Auswahl der Koordinatoren implementiert werden. Für Verfahren, die auf dem Austausch von Nachbarschaftslisten basieren gilt, dass sie ein hohes Verkehrsaufkommen bei den Knoten erzeugen, da keine Filterung vorgenommen wird. Für Verfahren mit gegenseitiger Benachrichtigung gilt, dass Knoten verantwortlich für die Nachbarschaftserkennung fremder Knoten sind. Dazu muss die Nachbarschaft jedes Nachbarn ständig überwacht und ausgewertet werden. Das kann bei einer hohen Knotendichte zu Problemen bei leistungsschwachen Knoten führen. Alle Verfahren haben gemeinsam, dass sie senderorientierte Verfahren sind. Das kann zu Kommunikationsoverhad führen, wenn Sender und Empfänger eine unterschiedliche AOI haben.

## 2.6 Fazit und offene Fragen

In diesem Kapitel wurden die grundlegenden Komponenten, die für eine hochskalierbare verteilte virtuelle Welt benötigt werden, vorgestellt. Obwohl einzelnen Detaillösungen und Mechanismen klar definiert sind, ist jedoch offen, welche Leistungsfähigkeit ein vollständiges Overlaynetzwerk erbringen kann. Dazu fehlt eine Möglichkeit, die Qualität von unterschiedlichen Verfahren in Relation zu setzen. Weiterhin lässt sich beobachten, dass alle vorgestellten Verfahren eine senderorientierten Ansatz verwenden. Das kann jedoch mit Overhead verbunden sein und im schlimmsten Fall sogar zu Netzwerkinkonsistenzen

führen.

Somit stellen sich die Fragen:

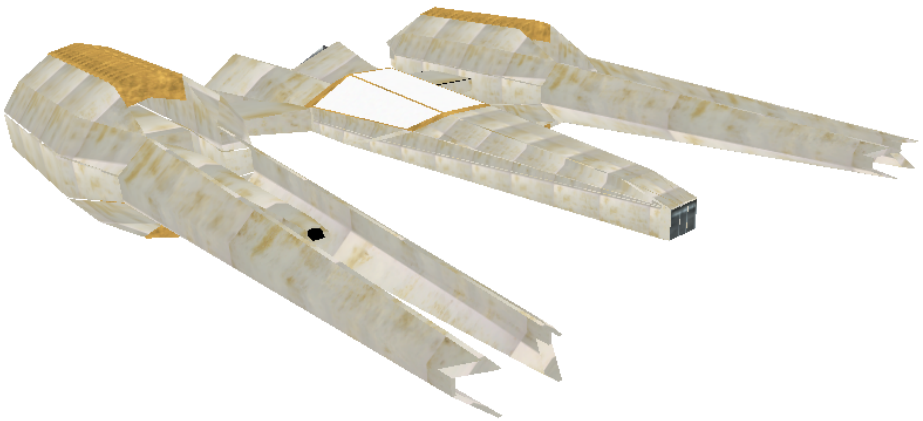
- Ist es möglich, einen empfängerbasierten (publish-/subscribe) Ansatz zu nutzen?
- Wie kann allgemein die Leistungsfähigkeit einzelner Ansätze gemessen und miteinander verglichen werden?

Darüber hinaus ist ebenso ungeklärt, wie sich ein derartiges Overlay verhält, wenn nicht nur Updatenachrichten zwischen den Knoten ausgetauscht werden, die eine sehr geringe Datenrate erzeugen, sondern wenn zusätzlich Bild- oder Sprachinformationen übertragen werden soll.

Diese Fragen können nicht theoretisch beantwortet werden; um eine zuverlässige Antwort geben zu können, ist eine reale Erprobung notwendig. Dafür werden im folgenden Kapitel die zentralen Komponenten der prototypischen Spielanwendung *Planet PI4* vorgestellt, die wir zu diesem Zweck entwickelt haben.

# Kapitel 3

# Das Netzwerkspiel Planet PI4



## 3.1 Problemstellung

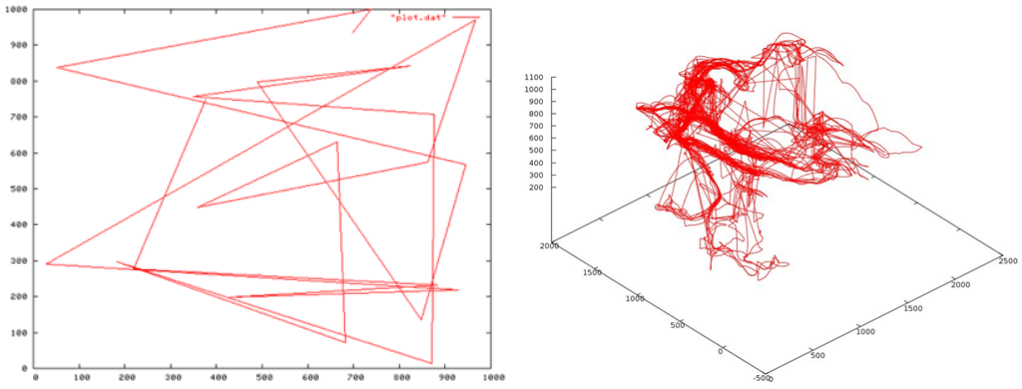
Die zentrale Komponente eines Onlinespiels ist der Austausch von Statusnachrichten. Durch das Kommunizieren von Bewegungen und Interaktionen der Spieler wird der Eindruck einer gemeinsam genutzten virtuellen Welt erzeugt. Welche Nachrichten an welchen Spieler übertragen werden, hängt jeweils vom aktuellen Zustand der Spieler und ihrer Umgebung ab. Die Struktur des Netzwerkes basiert auf Echtzeitinformationen der virtuellen Welt und muss daher kontinuierlich verändert werden. Damit unterscheiden sich virtuelle Welten von den meisten Netzwerkanwendungen, die eine statische Architektur verwenden. Das ist insbesondere für den Entwurf und die Erprobung von Netzwerkarchi-

tekturen relevant; eine Simulation der dynamischen Bedingungen ist zwingend notwendig.

Die einfachste Methode zur künstlichen Erzeugung von dynamischen Spielzuständen besteht in der Verwendung von Bewegungsmodellen. So können beispielsweise zufällige Bewegungen eines Spielers mit einem *Random-Waypoint-Modell* (RWP) simuliert werden (siehe Abbildung 3.1a). Ein derartiges Modell lässt sich mit wenigen Zeilen Code erzeugen und kann für einfache Netzwerktests eingesetzt werden. Dennoch ist es für unsere Zwecke nicht ausreichend, da eine Anbindung an eine Spielwelt fehlt. Das hat zur Folge, dass keine (oder nur begrenzte) echte Interaktionen simuliert werden können, welche allerdings einen Großteil der Netzwerknachrichten ausmachen. So können bei einem Shooter leicht 20 Schüsse und damit Netzwerknachrichten pro Sekunde entstehen. Jeder Treffer erzeugt ebenfalls Netzwerknachrichten. Die Entscheidung, ob ein Schuss ein Schiff trifft, erfordert jedoch viele Informationen aus der virtuellen Welt, etwa über die Richtung des Schusses, die Geschwindigkeit und das Ausmaß des Projektils und die Position und die Größe und Form des beschossenen Schiffes.

Eine weitere Möglichkeit zur Simulation dynamischer Bedingungen besteht darin, Aktionen von realen Spielern aufzuzeichnen (siehe Abbildung 3.1b) und anhand der gemessenen Interaktionssequenzen Spielsituationen zu erzeugen. Der Vorteil dieser *Traces* besteht darin, dass sie realistische Informationen über das Verhalten der Spieler wiedergeben. Sie sind jedoch auf eine konkrete Spielsituation und eine feste Anzahl an Spielern beschränkt und können daher nicht für Skalierbarkeitsuntersuchungen eingesetzt werden. Zudem enthalten sie gewaltige Datenmengen, deren Handhabung mit großem Aufwand verbunden ist.

Daher ist es für die Generierung einer realistischen und dynamischen Testumgebung sinnvoll, die Implementierung einer virtuellen Welt zu erstellen. So können sämtliche Spielsituationen mit beliebigen Spieleranzahlen simuliert werden.




(a) Bewegungssimulation mit dem Random Waypoint Modell

(b) Trace eines Quake3-Spielers im dreidimensionalen Raum

Abbildung 3.1: Bewegungen einer virtuellen Welt können anhand von Bewegungsmodellen simuliert werden (links) oder von einer realen Spielsession aufgezeichnet werden (rechts).

## 3.2 Implementierung

Als Grundlage für die in dieser Arbeit entwickelten Algorithmen und Mechanismen wurde die virtuelle Spielumgebung *Planet PI4* entworfen und implementiert. Planet PI4 beinhaltet sowohl ein vollständig funktionsfähiges third-person-Weltraum-Shooter-Spiel als auch eine Simulationsumgebung, die es ermöglicht, sämtliche Spielereignisse unter kontrollierten Bedingungen zu erzeugen und auszuwerten. Die Entscheidung für das Shooter-Genre ist dadurch begründet, dass Shooter die höchsten Echtzeitanforderungen haben. Spieler sind bei kleinen, schnellen Objekten, wie beispielsweise Projektilen, besonders sensibel gegenüber Verzögerungen. Die Weltraumumgebung wurde aus Skalierbarkeitsgründen gewählt. Dies ist notwendig, da besonders das Verhalten eines Netzes bei großen Teilnehmerzahlen von Interesse ist. Die Generierung einer virtuellen Umgebung für viele tausend Teilnehmer ist jedoch in der Regel mit großem Aufwand verbunden. Der Weltraum im Spiel hat dagegen eine theoretisch unendliche Ausdehnung, die nur von der Hardware begrenzt wird. Planeten und Asteroiden können über einen *pseudorandom number generator* (PRNG) automatisch und reproduzierbar erzeugt werden. Somit ist es mit Planet PI4 theoretisch möglich, Overlays mit unbegrenzt vielen Spielern zu simulieren bzw. unbegrenzt viele reale Spieler gleichzeitig in das Spiel zu inte-

grieren.

Aufgrund der umfangreichen Anforderungen haben an dem Spiel Planet PI4 insgesamt zehn Personen[1] mitentwickelt. Die dabei entstandene Simulations- und Evaluationsumgebung wurde intensiv in den Forschungsprojekten *Spontane virtuelle Netze* (SpoVNet) [13] und *Qualität von Peer-to-Peer-Netzwerken* (QuaP2P) [1] eingesetzt. Weiterhin steht die Implementierung der wissenschaftlichen Gemeinschaft für Forschungsarbeiten frei zur Verfügung. Der Aufbau der Software ist dabei von zwei elementaren Eigenschaften geprägt: Erstens können durch die Verwendung einer speziellen *Task-Engine* sämtliche Ereignisse, wie zum Beispiel Bewegungen, Schüsse oder die Einnahme einer Basis, sowohl in Echtzeit als auch in einer diskreten Eventsimulation erzeugt werden. Zweitens wurde ein modulares Netzwerkinterface entwickelt, das es ermöglicht, unterschiedliche Netzwerke einzusetzen, ohne dabei auf die eigentliche Spielimplementierung zugreifen zu müssen. Um den Aufbau der Software zu verdeutlichen, werden im Folgenden die verwendeten Bibliotheken und die Organisation der Software vorgestellt.

### 3.2.1 Bibliotheken

Die gesamte Softwareumgebung von Planet PI4 wurde plattformunabhängig in C++ entwickelt. Dabei wurden für verschiedene Bereiche quelloffene Bibliotheken verwendet.

**Irrlicht-Engine:** Die Irrlicht-Grafikengine ist eine open-source-3D-Bibliothek. Sie ist plattformunabhängig und stellt einen Direct3D, einen OpenGL und einen Softwarerenderer bereit. Viele Funktionen, die zur Generierung einer hochwertigen virtuellen Welt notwendig sind, können dabei genutzt werden. So gibt es Klassen zur Erstellung von Skyboxen/Skydomes, Terrainrenderer zur Erstellung von Landschaften und eine Vielzahl von Partikeleffekten zur Erzeugung von Explosionen oder zur Simulation von Feuer oder Flüssigkeiten. Darüber hinaus gibt es primitive Funktionen zur Simulation physischer Eigenschaften, wie Schwerkraft oder Kollisionserkennung.

[1] Benjamin Guthier, Max Lehn, Robert Rehner, Dimitri Wulffert, Denis Lapiner, Damian Czarny, Philip Mildner, Philipp Schaber, Britta Weber und der Autor


Abbildung 3.2: Screenshot des Spiels Planet PI4

**Boost:** Boost ist eine vielseitig einsetzbare C++-Bibliothek. Sie enthält mathematische und numerische Funktionen sowie Implementierungen, die Nebenläufigkeit und Multithreading unterstützen. Zusätzlich stellt Boost viele nützliche Container- und Iteratorklassen bereit.

**Log4cplus:** Log4cplus ist eine umfangreiche Logging- und Trackingbibliothek. Sie wurde besonders intensiv eingesetzt, um Zustände und Zustandsänderungen während der Testläufe oder Simulationen zu ermitteln und zu speichern.

**sqlite:** Sqlite stellt ein Datenbankinterface bereit, das sowohl für die persistente Speicherung von Spieleinformationen als auch für das Speichern der Evaluationsinformationen genutzt wurde.

### 3.2.2 Spielaufbau

Der Aufbau des Spiels beinhaltet verschiedene Spielmodi. Allgemein navigieren die Spieler ein Raumschiff durch ein Asteroidenfeld. Dabei können sie entweder in einem Deathmatch-Modus einzeln oder gemeinsam in unterschiedlichen Teams gegeneinander spielen. Das Ziel ist es, andere Spieler bzw. Teams zu zer-

stören und strategische Punkte einzunehmen. Zu den strategischen Punkten gehören Basen und Reparaturpunkte, die unterschiedliche Boni, wie zusätzliche Energie oder Regeneration der Lebenspunkte, gewähren. Um einen derartigen *Point of Interest* (POI) einnehmen oder nutzen zu können, ist es notwendig, sich in der unmittelbaren Umgebung des POI aufzuhalten, ohne dass sich dabei ein gegnerischer Spieler in der Zone aufhält. Die POI wurden verwendet, da sie ein typisches Element von virtuellen Welten darstellen. Sie erzeugen eine Ansammlung von Spielern, wie sie in Städten, Handelsplätzen oder Schlachten entstehen können. Diese Verdichtungen stellen besondere Anforderungen an ein Netzwerk, da mit ihnen ein hohes Aufkommen von Nachrichten verbunden ist. Abbildung 3.2 zeigt einen Screenshot des Spiels.

### 3.2.3 Komponenten

Im Folgenden wird eine Übersicht über die implementierten Komponenten von Planet PI4 gegeben. Insgesamt lassen sich die über 200 Klassen in acht Themenbereiche unterteilen:

**KI:** Die KI-Komponente enthält sämtliche Klassen, die zur autonomen Steuerung von Spielern verwendet werden. Dazu gehören ein Interface und drei Implementierungen. Konkret wurden verschiedene skriptbasierte und außerdem eine zustandsorientierte und eine zielorientierte künstliche Intelligenz (KI) entwickelt. Damit verbunden sind Klassen für Zustände und Ziele und die entsprechenden logischen Strukturen einer *Finite State Mashine* (FSM) und eines *Behavior Trees* (BT).

**Frontend:** Das Frontend umfasst die grafische Darstellung der Benutzeroberfläche. Das beinhaltet sowohl die Szene mit der dazugehörenden Kamera und der Beleuchtung als auch die Skybox und das *Head Up Display* (HUD). Dieses besteht hauptsächlich aus einem Radar, einer Statusanzeige für Schilde und der Anzeige allgemeiner Statusnachrichten. Zusätzlich werden die Tastaturbelegungen für die Schiffssteuerung festgelegt.

**Game:** Diese Komponente enthält die Verwaltung der Spielwelt. Dazu gehören sowohl sämtliche passiven Objekte wie Asteroiden und Schiffe als auch die aktiven Objekte, deren Zustandsverwaltung über das Netzwerk gesteuert wird, wie Basen oder Reparaturpunkte.

**Mobility Model:** Sowohl für den Simulationsmodus als auch für verschiedene KI-Implementierungen werden Bewegungungsmodelle verwendet. Dafür wurden ein Interface und verschiedene Implementierungen (Random Waypoint/ Random Point of Interest, Schwarm-Modell) erstellt.

**Network:** Zu diesem Themenbereich gehören sämtliche Klassen der Netzwerkengine. Dazu gehört die Implementierung eines Transprotprotokolls (CUSP), ein spezieller Channel-Publish-Subscribe-Dienst, der sowohl für Teamkommunikation als auch das Verwalten der aktiven Objekte eingesetzt wird, und verschiedene Netzwerkcontainer und Interfaces, die als Schnittstelle von Overlayimplementierungen genutzt werden können. Zusätzlich gibt es sechs verschiedene Implementierungen einer Overlaystruktur für das Netz. Diese werden im folgenden Abschnitt genauer beschrieben.

**Statistics:** Die Statistikkomponente enthält sämtliche Klassen zur Verfolgung und Messung von Daten und zur Erstellung von Statistiken.

**System:** Die Systemklassen steuern die Laufzeitumgebung von Planet PI4. Dazu gehören verschiedene Zufallsgeneratoren sowie die zentrale Komponente zur Steuerung aller Ereignisse, der *Task Engine*

**Utilities:** Die letzte Komponente enthält mathematische Hilfsfunktionen, Systemparameter, Container und Iteratorenklassen.

### 3.2.4 Netzwerkimplementierungen

Da Planet PI4 von Beginn an von unterschiedlichen Projekten für Netzwerkevaluationen genutzt wurde, wurden Netzwerkinterfaces für den Auf- und Abbau von Netzwerkverbindungen, das Interest Management und die Verbreitung von Spieleereignissen (GED) implementiert. Diese können genutzt werden, um eine Overlaystruktur zu implementieren, ohne dabei weiteres Wissen über die Implementierung von Planet PI4 haben zu müssen. Zusätzlich ist es möglich, zwischen verschiedenen Implementierungen zu wechseln (siehe [61]). Insgesamt wurden so sechs unterschiedliche Implementierungen eines Overlays in Planet PI4 integriert. Dazu gehören eine Client-/Server-Version, drei klassische P2P-Overlay-Implementierungen und zwei Konzepte, die auf einer Verwendung wei-

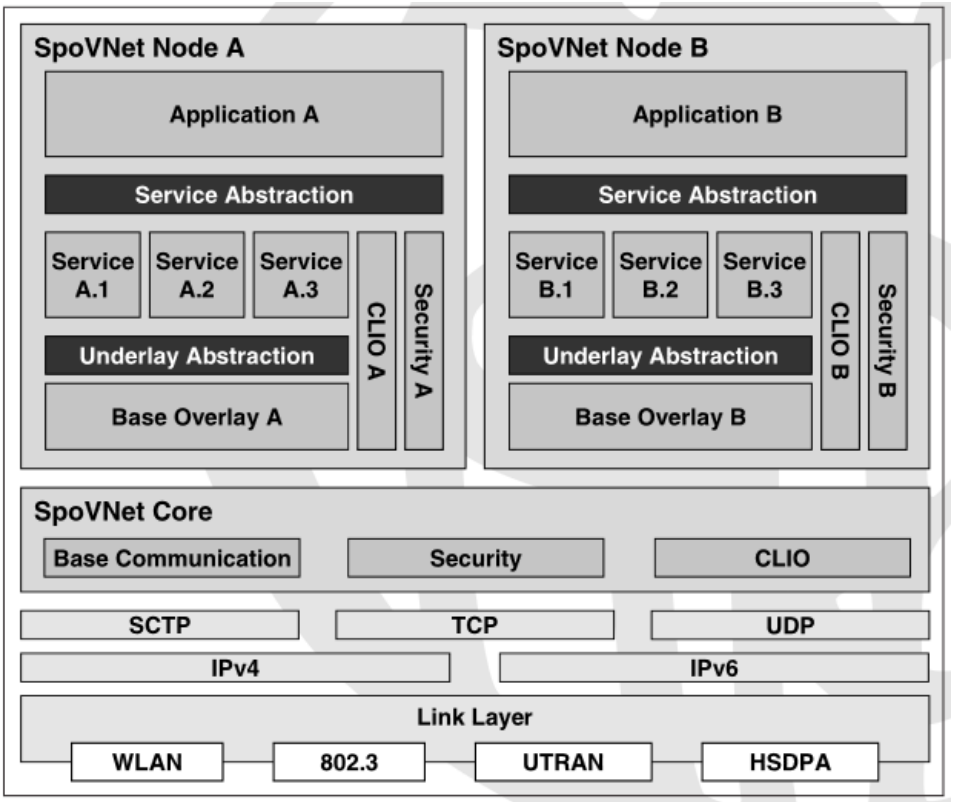



Abbildung 3.3: Übersicht über die SpovNet-Architektur. (Bildquelle: [63])

terer Netzwerkdienste wie Skype oder der SpoVNet Middleware basieren. Zu den P2P-Overlays gehören zum einen der im folgenden Kapitel vorgestellte Ansatz, zum anderen zwei P2P-Netzwerke, die am Lehrstuhl für *Databases and Distributed Systems (DVS)* in Darmstadt entwickelt wurden, *pSense* [49] und Bubblestorm [57, 58]. Die beiden dienstbasierten Konzepte werden im Folgenden beschrieben:

### SpoVNet

Im Rahmen des BWFit-geförderten Projektes Spontane virtuelle Netzwerke (SpoVNet)[39] wurde erforscht, wie sich Netzwerkdienste an dynamisch verändernde Netzwerkanforderung anpassen können. Dabei gilt es, die Ursachen von Veränderungen (z.B., Veränderung der Zugangstechnologie) vor den Nutzern zu verbergen und eine durchgehende Dienstgüte bereitzustellen. Realisiert wird dies durch ein intelligentes Verwalten von Ressourcen und ein Zusammenspiel

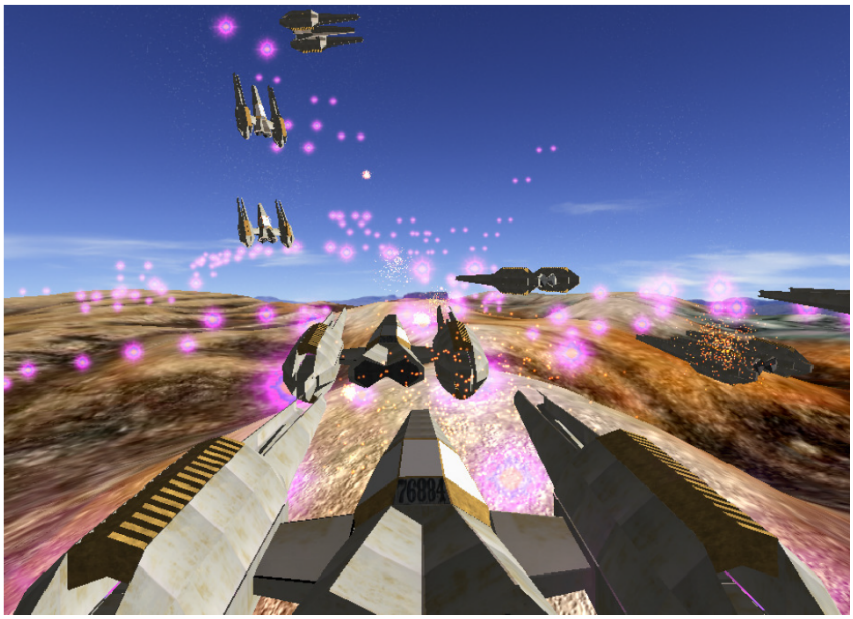


Abbildung 3.4: Screenshot des SpovNet-Demonstrators 2009 in Mannheim

mehrerer leistungsfähiger Netzwerkdienste. Abbildung 3.3[2] zeigt eine Übersicht über die einzelnen SpoVNet-Komponenten, die gemeinsam mit Planet PI4 in einen lauffähigen Demonstrator integriert wurden. Im einzelnen wurden dabei folgende Dienste eingesetzt:

**Ariba:** Ariba [25] ist eine Middleware, die die Basiskommunikation in einem SpovNet bereitstellt. Dazu verwendet Ariba ein Peer-to-Peer-Netzwerk für jede Anwendung, die es benutzt. Dabei ist das Overlay in der Lage sich vollständig selbst zu organisieren. Zusätzlich kann sich die Struktur des Overlays an verändernde Netzwerkbedingungen anpassen. Dazu werden zwei unterschiedliche Abstraktionsebenen eingesetzt. In der Abbildung 3.3 sind diese innerhalb der beiden Knoten A und B zu erkennen. Die erste Abstraktionseben bewirkt ein Verbergen der Underlayparameter. Umgesetzt wird diese Abstraktion von dem SpoVNet Core. D.h., der SpoVNet Core (implementiert von Ariba) abstrahiert von der verwendeten Zugangstechnologien und den Netzwerkprotokollen und stellt einem SpoVNet-Knoten lediglich eine Overlayadresse zur Verfügung. Die

[2]Die SpovNet-Architektur wurde im Laufe der Zeit mehrfach abgewandelt. Die Übersichtsgrafik zeigt die Architektur aus der Phase der Zusammenarbeit von SpoVNet und Planet PI4 (2009)

zweite Abstraktion wir innerhalb eines Knotens umgesetzt und bewirkt, ein Verbergen der Netzwerkdienste vor der Anwendung. Somit kann sich eine Anwendung darauf beschränken, lediglich Daten an eine Adresse zu schicken und Dienstgüteanforderungen anzumelden. Ariba wurde am Lehrstuhl für Telematik in Karlsruhe entwickelt.

**CLIO / UNISONO:** Um den aktuellen Zustand des Netzwerkes, der Verbindungen und den Umfang der verfügbaren Ressourcen in Echtzeit ermitteln zu können, wurde der Crosslayer Informationsdienst CLIO (später UNISONO) [20, 44] entwickelt. Um ein gesamtes Bild der aktuellen Situation zu erhalten, ist der CLIO-Dienst in mehreren Schichten verankert. Dadurch ist er sowohl in der Lage, einfach Verbindungsparameter, wie zum Bespiel Paketverzögerungen, Paket-Verlustrate oder Bandbreite zu ermitteln, als auch Zustandsparameter der Overlays zu messen. Clio wurde am Lehrstuhl für Netzarchitekturen und Netzdienste in Tübingen entwickelt

**Cordies:** Cordies ist ein leistungsfähiger Ereignisdienst, der Netzwerkereignisse in einem SpovNet überwacht und entsprechende Ressourcen einer Anwendungen bereitstellt. Dabei ist er in der Lage, sogar logische Verküpfungen von Ereignissen zu ermitteln. Diese Funktion wurde von Planet PI4 genutzt, um Kommunikationspartner zu ermitteln. So ist Cordies in der Lage, die aktuelle Position eines Spielers, aus einer Positionsupdatenachricht auszulesen. Zusätzlich kann dem Ereignisdienst ein Parameter für die Größe der AOI eines Spielers übergeben werden und logische Bedingungen, die an ein Ereignis gestellt werden, festgelegt werden. Konkret kann so die Spieleanwendung dem Ereignisdienst signalisieren: Wenn der Abstand eines Spielers X zu der eigenen Position Y kleiner wird, als der Radius der AOI ($D(X,Y) < Radius$), löse das Ereignis *Ankunft eines Knotens* aus, wird der Abstand größer wird, das Ereignis *Verlassen eines Knotens* ausgelöst. Auf den ersten Blick erscheint dieses Vorgehen als sehr spezialisiert, da der Ereignisdienst Kenntnis von sehr spezifischen Spieleinformationen haben muss. Tatsächliche wird diese Funktionalität lediglich durch einfache logische und numerische Verknüpfungen von Zahlen umgesetzt. D.h., Cordies sucht nach Kommunikationen im Netz, die

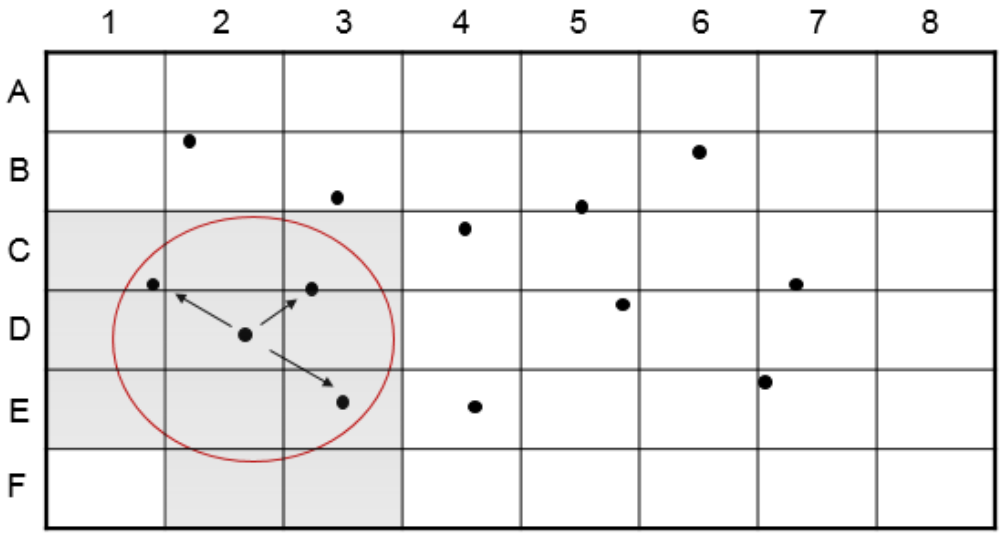



Abbildung 3.5: Zonenkonzept, das bei der Skype4Games-Implementierung verwendet wurde. Für jede Zone wurde eine Skypekonferenz genutzt. Spieler kommunizieren dabei mit allen Zonen, die ihre AOI schneiden.

ähnliche Parameter haben, ohne dabei zu wissen welche Bedeutung die Operationen haben. Die intelligente Nutzung der Information wird dann von der Anwendung vollzogen. Cordies wurde am Institut für Parallele und Verteilte Systeme (IPVS) in Stuttgart entwickelt und implementiert.

Das gesamte Zusammenwirken aller Komponenten wurde bei einer Demonstration am Lehrstuhl für Praktische Informatik IV in Mannheim 2009 vorgeführt (siehe Abbildung 3.4). Dabei kamen 14 Spieler an den vier Standorten Stuttgart, Karlsruhe, Tübingen und Mannheim zum Einsatz. Bei der Demonstration konnte live visualisiert werden, wie die Aribasoftware einzelne Knoten in das Netzwerk einfügt und entfernt. Zusätzlich wurden unterschiedliche Veränderungen am Netzwerk vorgenommen. Diese wurden von dem CLIO-Dienst erkannt, und das Netz war in der Lage, seine Architektur daraufhin anzupassen. Der Ereignisdienst Cordies wurde für die Erstellungen der Kommunikationsgruppen eingesetzt. Ankommende Spieler wurden mit einem rot eingefärbten Schiff visualisiert, wodurch die Echtzeitfähigkeit der Software gut erkennbar war.

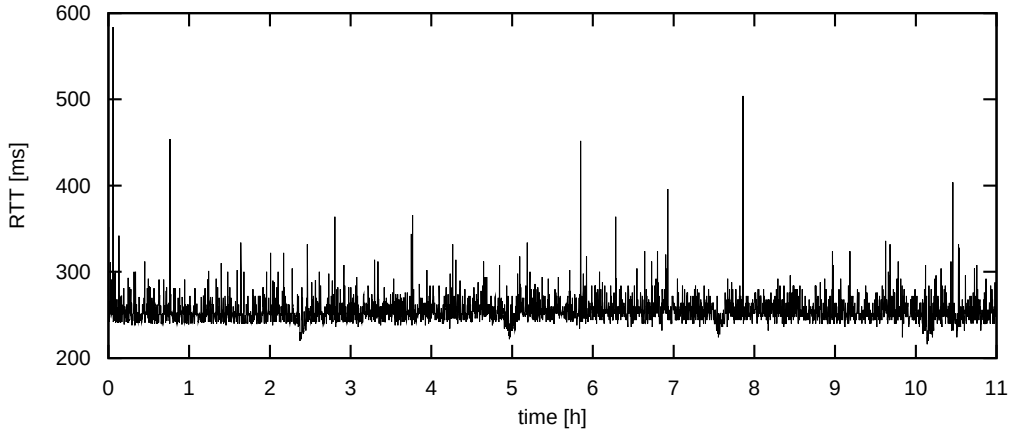



Abbildung 3.6: Langzeitmessung der Latenz einer Skypeverbindung zwischen Deutschland und Kanada.

### Skype4Games

Ein weiterer Ansatz, externe Netzwerkdienste als Netzwerkimplementierung zu nutzen, bestand in der Verwendung der Skype API [52, 59]. Skype ist eine proprietäre P2P-Software für Sprach- und Textkommunikation und daher auf echtzeitfähige Übertragung optimiert (vgl. [5]). Obwohl Skype nicht als quelloffene Software genutzt werden kann, ist es möglich, auf unterschiedliche Kommunikationsmechanismen anhand einer offenen Schnittstelle zuzugreifen. So kann beispielsweise auf Skypeadressen und Freundeslisten zugegriffen werden, um Textnachrichten zu verschicken. Das gesamte Management der Verbindungen, das den Auf- und Abbau inklusive NAT-Traversal und die Überwachung der Verbindungsqualität beinhaltet, kann dabei mitbenutzt werden. Zusätzlich können direkte Sprachverbindungen aufgebaut werden. Um diese Funktionalität in Planet PI4 einzusetzen, wurde ein primitiver IM-Dienst entwickelt, der die Spielwelt in ein Gitter aus statischen Zonen unterteilt (siehe Abbildung 3.5). Innerhalb einer Zone wurden dann Skypekonferenzen mit allen Teilnehmern aufgebaut. Sämtliche Updatenachrichten, wie Positionsveränderungen, Schüsse, Treffer oder Kills, wurden dann als Skypetextnachricht verschickt. Diese Form stellt zwar keine skalierbare Lösung dar, da Skype eine Obergrenze von 25 Teilnehmern für Kommunikationsgruppen festgelegt hat, sie konnte jedoch für unterschiedliche Messungen und Evaluationen eingesetzt werden. So wurde beispielsweise in einem Langzeittest über elf Stunden die

Stabilität der Verbindungen zwischen Deutschland, Kanada und Frankreich getestet. Abbildung 3.6 zeigt eine Latenzmessung der Verbindung zwischen Deutschland und Kanada. An der Stabilität und der Echtzeitfähigkeit der Verbindung ist erkennbar, dass Skype eine ideale Plattform für Onlinespiele darstellt. Für den Einsatz in einem MMOG wird jedoch noch ein geeignetes Verfahren für das Interest Management benötigt.

## 3.3 Fazit

Mit Planet PI4 wurde eine vielseitig einsetzbare Plattform implementiert, anhand derer sowohl Overlaynetzwerke, als auch assoziierte Netzwerkdienste entwickelt und evaluiert werden können. Sie wurde als Basis für alle Dienste eingesetzt, die im weiteren Verlauf dieser Arbeit vorgestellt werden.

# Kapitel 4

# Sprachkommunikation

## 4.1 Problemstellung

Sprache ist ein Medium, das in virtuellen Welten genutzt wird, um gruppenbasiertes Spielen oder die Bildung sozialer Gemeinschaften zu unterstützen. Mittels Sprache können Spieler schneller und einfacher kommunizieren als durch die Eingabe von Text. Dennoch ist in derzeitigen MMOG-Spielanwendungen meist keine oder nur eine unzureichende Umsetzung einer Sprachkommunikation integriert. Ursache dafür sind die hohen Anforderungen an das Netzwerk. Eine integrierte, serverbasierte Lösung erzeugt beispielsweise Datenraten, die das gesamte Verkehrsvolumen für das Betreiben einer virtuellen Welt (ein Ab-

schätzung dafür basiert auf Messungen von Svoboda et al.[54]) um ein Vielfaches übersteigen.

Um dennoch Sprache im Spiel nutzen zu können, greifen Spieler daher zu externen Lösungen, die unabhängig von der jeweiligen Spielanwendung operieren. Diese Nutzung einer zusätzlichen Software hat jedoch entscheidende Nachteile. So fehlt extern erzeugten Audioströmen die Anbindung an die Spiellogik. Effekte, die die Sprache in ihre virtuelle Umgebung einbetten, können daher nicht genutzt werden. Ebenso muss die Zusammenstellung von Gruppen manuell durchgeführt werden. Für kleine, statische Gruppen bedeutet dies keine starke Einschränkung. Bei schnell wechselnden Teilnehmern oder Gruppen mit unbekannten Spielern wird jedoch eine automatische Gruppenkommunikation notwendig. Außerdem ist die Nutzung von externer Software meist mit zusätzlichen Kosten verbunden.

Ziel dieses Kapitels ist es daher zu untersuchen, welche Schritte notwendig sind, um eine integrierte und skalierbare Sprachkommunikation umzusetzen. Es wird eine prototypische Implementierung vorgestellt, die auf einer hybriden P2P-Architektur basiert und die in das Spiel PlanetPI4 integriert wurde.

## 4.2 Anforderungen

Die Übertragung von Sprache erfordert eine intensive Nutzung von Netzwerk- und Prozessorressourcen. Um dennoch eine effiziente Funktionsweise zu ermöglichen, werden Anwendungen gezielt für konkrete Szenarien entworfen. So nutzt Skype beispielsweise ein P2P-Overlay, um eine globale Verbindung zu ermöglichen. Die Kommunikation selbst ist dabei für kleine Gruppen optimiert [1]. Aufgrund der verteilten Struktur von Skype kann es dabei zu Schwankungen der Sprachqualität oder zu Verbindungsverlusten kommen. Darüber hinaus erfordert das Management des P2P-Overlays zusätzliche Ressourcen, was besonders in Verbindung mit leistungsfordernden Spielen zu Problemen führen kann.

Lösungen wie Teamspeak, Ventrillo oder Adobe Connect nutzen einen speziellen Server, um bessere Dienstgüte und größere Gruppen ( bis zu 500 Teil-

[1] 2 bis 5 Teilnehmer - maximal sind 25 Teilnehmer möglich, allerdings sinkt die Qualität mit zunehmender Teilnehmerzahl

nehmern) gewährleisten zu können. Durch ihre Architektur ist auch ein effizienteres Ressourcenmanagement möglich. Daher werden sie häufig als externer Dienst in aktuellen MMOGs verwendet. Die entstehenden Datenraten müssen jedoch durch monatliche Nutzungsgebühren bezahlt werden.

Für eine realistische Sprachübertragung in Spielen haben Boustead et al. eine Client-/Server-Lösung entwickelt [9, 42, 47]. Sie haben die Audioströme der Spieler mit der Spielwelt in Verbindung gebracht indem sie Umgebungseffekte wie Hall oder Dämpfung und die Orientierung des Sprechers in den Sprachstrom integrierten. Auf diese Art repräsentiert jeder Spieler eine persönliche Mischung seiner Sprache und Umgebung. Aufgrund der hohen Bandbreitenanforderungen haben sich Boustead et al. gegen die Nutzung eines P2P-Overlays entschieden. Stattdessen wurde eine naive Client-/Server-Implementierung durch die Nutzung von *Angular Clustering* verbessert. Dabei werden Audioströme, die aus dem selben Richtungsabschnitt kommen, von einem Server zusammengefasst. Somit reduziert sich die Anzahl der eingehenden Audioströme eines Knotens auf die Anzahl der Richtungsabschnitte. Dennoch wird für das Verwalten und Mischen aller Audioströme ein zentraler Server benötigt. Somit ist diese Lösung für den Einsatz in MMOGs ungeeignet.

Um nun gezielt eine Lösung für MMOGs entwerfen zu können, müssen zunächst die konkreten Anforderungen ermittelt werden. Dafür werden in den folgenden Abschnitten Sprachqualität, Skalierbarkeit, Gruppengröße, Verzögerungstoleranz und Immersiveness der Anwendung diskutiert.

### Sprachqualität

Die Sprachqualität ist ein wichtiger Faktor der Leistungsfähigkeit eines Sprachkommunikationssystems. Die Übermittlung von Sprache ist nur dann sinnvoll, wenn sich die Teilnehmer gegenseitig verstehen können. Objektiv kann die Qualität durch die *Perceptual Evaluation of Speech Quality* (PESQ) gemessen werden. Dieser Wert kann durch einen subjektives Maß wie den *Mean Opinion Score* (MOS)[2] abgebildet werden. Der MOS reicht von Qualität 1 (schlecht) bis zu 5 (exzellent). Ein Wert von 3 ermöglicht das Verstehen von Sprache mit leichtem Aufwand. Ein MOS von 4 bedeutet, dass Sprache ohne Probleme verstanden werden kann.

Allgemein lässt sich eine höhere Sprachqualität dadurch erreichen, dass für

| Codec | Datenrate in kbit/s | MOS |
|---|---|---|
| G.711 | 64 | 4,1 |
| iLBC | 15,2 | 4,14 |
| AMR | 12,2 | 4,14 |
| G.729 | 8 | 3,92 |
| GSM EFR | 12,2 | 3,8 |

Tabelle 4.1: Übersicht über den erzielten MOS verschiedener Sprachkodierungen bei unterschiedlichen Datenraten.

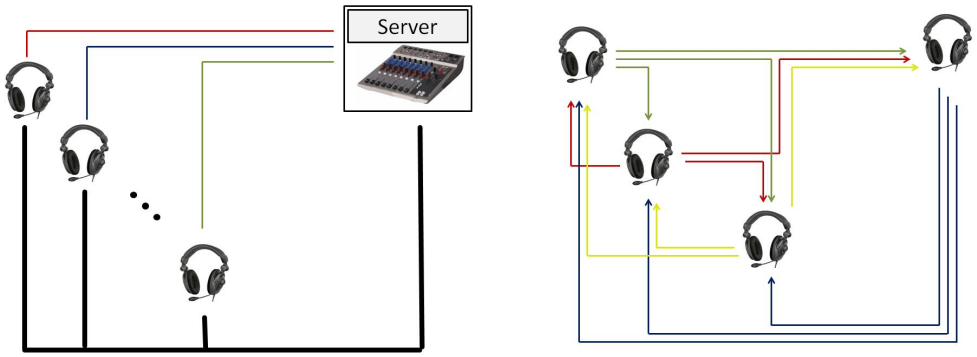
die Kodierung der Sprache eine höhere Bitrate verwendet wird. Jedoch ist damit auch eine höhere Datenrate für die Kommunikation verbunden. Besonders bei verteilten Anwendungen muss daher die Bitrate sinnvoll gewählt werden. Tabelle 4.1 zeigt eine Übersicht über die Datenrate und den MOS-Wert unterschiedlicher Sprachkodierungsverfahren [2].

Es ist zu erkennen, dass der *GSM Full Rate Standard* im Vergleich zu moderneren Standards den niedrigsten MOS-Wert erzielt. Dennoch wurde er über viele Jahre als Kommunikationsstandard für mobile Sprachkommunikation genutzt. Daher ist es sinnvoll, einen MOS von 3,5 als Minimalanforderung für eine globale MMOG-Sprachkommunikation festzulegen. Beim Einsatz eines Codecs mit variabler Bitrate kann also bei Bedarf die Bitrate reduziert werden, solange ein MOS von 3,5 erreicht wird.

### Skalierbarkeit - Gruppengröße

Die Besonderheit einer Sprachanwendung für MMOGs besteht in ihrer Skalierbarkeit. Es gilt dabei, zwei Dimensionen dieser Skalierung zu gewährleis-

[2] G.711 ist der von ISDN verwendete Codec, iLBC (Internet Low Bitrate Codec) ist ein quelloffener Codec, der für VoIP-Kommunikation entworfen wurde, ARM (Adaptive Multirate) wird im Mobilfunk eingesetzt, G.729 ist ein ITUT Standard des Conjugate Structure Algebraic Code Excited Linear Prediction (CS-ACELP) Verfahrens, GSM EFR (Global System for Mobile Communications Enhanced Full Rate Codec) ist ein Mobilfunkstandard.




(a) Client-/Server-Architektur (b) Peer-to-Peer-Architektur

Abbildung 4.1: Bei einer Client-/Sever-Architektur (links) müssen die Clients nur einen ausgehenden Audiostrom versenden. Der Server muss dagegen die Ströme aller Teilnehmer mischen. Bei der Peer-to-Peer-Architektur (rechts) müssen die Teilnehmer Audioverbindungen zu allen anderen Teilnehmern in Hörreichweite aufbauen und die eingehenden Ströme selbst mischen.

ten: Gruppengröße und Gesamtteilnehmerzahl. Eine Skalierbarkeit in beiden Dimensionen lässt sich jedoch weder unabhängig voneinander noch unbegrenzt erreichen. Ursache dafür ist, dass zur Skalierung der beiden Dimensionen jeweils eine andere Architektur besser geeignet ist.

So hat eine Client-/Server-Architektur den Vorteil, dass jeder Client seinen eigenen Audiostrom lediglich an den Server schicken muss. Im Gegenzug erhalten die Clients vom Server einen vollständig gemischten Audiostrom (siehe Abbildung 4.1a). Dabei entsteht bei den Clients ein Datenvolumen von jeweils einem Audiostrom für den Sende- und den Empfangsweg. Vom Server müssen hingegen bei $N$ Teilenehmern $N$ eingehende und $N$ ausgehende Audioströme verarbeitet werden. D.h., die Gesamtteilnehmerzahl eines Client-/Serversystems wird von der Leistungsfähigkeit des Servers limitiert. Die Gruppengröße spielt dabei keine Rolle, da sie stets $\leq$ der Gesamtteilnehmerzahl ist und sich die Datenraten bei unterschiedlichen Gruppengrößen nicht verändern.

Somit ist es für Spiele mit kleinen virtuellen Welten, wie beispielsweise Counterstrike, Starcraft oder Battlefield, möglich, die Sprachkommunikation vollständig von einem Server verwalten zu lassen. Selbst bei einer maximalen Teilnehmerzahl (die maximale Teamgröße von Battlefield liegt beispielsweise bei 64 Spielern) müssten jeweils nur 128 ein- und ausgehende Verbindun-

gen übertragen und verarbeitet werden. Das würde bei der Verwendung eines Speex-Codec mit höchster Bitrate (Worstcase-Abschätzung) zu einem Bandbreitenverbrauch von jeweils

$$128 \cdot 44\ kbit/s = 5632\ kbit/s$$

für den Sende- und den Empfangsweg beim Server führen. Dieses Datenaufkommen ist zwar beträchtlich, kann aber mit aktuellen Datenverbindungen bewerkstelligt werden. Hingegen für MMOGs wie World of Warcraft, oder Eve Online mit über 10.000 gleichzeitigen Teilnehmern ist eine Client-/Server-Lösung derzeit nicht effizient möglich.

Im Gegensatz dazu haben P2P-Lösungen gezeigt, dass sie eine Kommunikation von mehreren Millionen Teilnehmern ermöglichen können (Skype) [52]. Grund dafür ist, dass die Audioströme nicht über einen zentralen Server, sondern direkt zwischen den Clients versendet werden. Das bedeutet, dass das System unabhängig von der Gesamtteilnehmerzahl operieren kann. Jedoch spielt die Größe der Kommunikationsgruppen eine wichtige Rolle, da jeder Teilnehmer an jedes Mitglied der Gruppe einen separaten Audiostrom senden muss (vgl. Abbildung 4.1b). Ebenso muss von jedem Mitglied ein Audiostrom empfangen werden. Zusätzlich müssen alle eingehenden Audioströme zu einem einzigen gemischt werden.

Es ist zu erkennen, dass eine allgemeine Lösung, die weder von der Teilnehmerzahl noch von der Gruppengröße abhängt, unmöglich ist. Um dennoch eine effiziente Lösung zu finden, gilt es, die Systemarchitektur bestmöglich an die tatsächlichen Anforderungen von Gruppengröße und Gesamtteilnehmerzahl anzupassen.

Die mögliche Gesamtteilnehmerzahl sollte dabei mindestens die Anzahl der gleichzeitigen Teilnehmer einer virtuellen Welt abdecken. Als Richtlinie können dafür 50.000 Teilnehmer angenommen werden. Dieser Wert orientiert sich an den Teilnehmerzahlen von Eve Online (siehe Abbildung 2.6). Um eine Einschätzung der auftretenden Gruppengrößen zu bekommen, müssen typische Verhaltensmuster der Spieler betrachtet werden. In aktuellen MMOGs wird dabei Sprachkommunikation hauptsächlich in zwei Szenarien verwendet: kooperatives Spielen und soziale Kommunikation. Im ersten Fall versucht eine Gruppe von Spielern, eine Aufgabe, die für einzelne Spieler zu anspruchsvoll

ist, gemeinsam zu erledigen. Dabei erfordert effizientes Teamplay die Koordination der einzelnen Spieler. Erfahrene Spieler nehmen dazu die Rolle eines Mentors an und leiten die Gruppe durch Kommandos. Der Rest der Gruppe beschränkt sich darauf, den Kommandos zuzuhören. Nur gelegentlich werden Statusmeldungen durchgegeben. Für gewöhnlich bestehen diese Gruppen aus einer Größe von 5 bis 25 Spielern. Größere Gruppen erfordern zusätzlichen logistischen Aufwand für die Zusammenstellung und Organisation. Darüber hinaus wird eine hierarchische Befehls- bzw. Kommunikationsstruktur notwendig. Derart große Gruppen werden zwar von einigen MMOGs unterstützt, es hat sich jedoch gezeigt, dass einem Großteil der Spieler der Aufwand dafür zu hoch ist und sie daher kleinere Gruppen bevorzugen.

Die zweite Form der Kommunikation in MMOGs ist die soziale Kommunikation. Dabei unterhalten sich befreundete Spieler während der Zeiten, in denen es keine aktiven Gruppenereignisse gibt. Es werden Erfahrungen und Spielerlebnisse ausgetauscht. D.h., es findet eine symmetrische Kommunikation statt. Die Größe dieser Gruppen ist ebenfalls variabel, dennoch gibt es eine implizite Einschränkung, da gerade bei symmetrischer Kommunikation eine gewisse Sprachdisziplin erforderlich ist. Sprechen zu viele Personen gleichzeitig, ist es nicht mehr möglich, sinnvoll zu kommunizieren. Dieser Effekt ist bereits bei zehn Personen deutlich zu erkennen. Daher kann die maximale Größe sowohl von Spielgruppen als auch von sozialen Gruppen auf 25 festgelegt werden.

Zusätzlich besteht die Möglichkeit von offenen Kommunikationsgruppen, d.h. Gruppen, deren Teilnehmerzahl unbekannt ist. Diese können beispielsweise auf einem virtuellen Marktplatz auftreten, auf dem sich Spieler treffen, um miteinander zu handeln. Für diese Gruppen gibt es keine explizite Einschränkung. Daher muss die zulässige Maximalgröße abhängig von den verfügbaren Ressourcen dynamisch festgelegt werden.

Zusammenfassend lässt sich festhalten, dass eine Anwendung gesucht ist, die eine Sprachkommunikation für mindestens 50.000 Gesamtteilnehmer ermöglicht und dabei Gruppen von bis zu 25 Teilnehmern unterstützt.

#### Netzwerkverzögerungen

Menschen sind in der Lage, bereits geringe Verzögerungen bei der Übertragung von Sprache wahrzunehmen (eine Abschätzung liefert Liang et al. [37]). Verzö-

gerungen von wenigen hundert Millisekunden werden in einem Dialog schon als störend empfunden. Dieser Umstand muss beim Design eines Sprachkommunikationssystems in Betracht gezogen werden. So können beispielsweise Konzepte wie Application Layer Multicast (ALM) in der Regel nicht verwendet werden. Allgemein sind P2P-Anwendungen in Bezug auf Verzögerungen einer Client-Server-Anwendung überlegen, da sie direkte Verbindungen nutzen können. Jedoch können auch Client-/Server-Implementierungen für die Sprachkommunikation eingesetzt werden, da sie selbst in einem Worstcaseszenario (z.B. zwei deutsche Clients und ein Server, der sich in Neuseeland befindet) noch in der Lage sind, die Sprache schnell genug zu übertragen (RTT Deutschland - Neuseeland $\approx$ 300 ms).

Im Falle der MMOG-Sprachkommunikation spielt eine zweite Verzögerung eine wichtige Rolle: die Zeit, die benötigt wird, um einen Kommunikationspartner zu ermitteln und eine Verbindung zu diesem aufzubauen. Bei einer reinen P2P-Kommunikation kann es durchaus einige Sekunden dauern, bis eine Verbindung einsatzbereit ist, da Prozesse, wie die Ermittlung der Nachbarn, NAT-Traversal oder das Aushandeln der Verbindungsparameter, zeitkritische Operationen sein können. Dieser Umstand kann umgangen werden, wenn das Kommunikationssystem in der Lage ist, zukünftige Verbindungen vorauszusehen und den Verbindungsaufbau bereits zu initiieren, bevor er benötigt wird.

### Immersiveness

Der Begriff Immersiveness bezeichnet im Zusammenhang mit MMOGs die Fähigkeit eines Spiels, Spieler in das Geschehen der virtuellen Welt hineinzuziehen. Dazu können Mittel wie Tiefe der Story, Realismus der Grafik oder die Gestaltung der Charaktere genutzt werden. Zusätzlich kann auch die Übertragung der Sprache für diesen Effekt eingesetzt werden. Basis dafür ist das Verknüpfen der Spiellogik mit der Übertragung und Verarbeitung der Sprache. Information aus der virtuellen Welt, wie die relativen Positionen von Sprechern und Zuhörern oder deren Umgebung, können mit in die Kodierung der Sprache einbezogen werden. Konkret wird dazu die Lautstärke eines Sprechers gedämpft, wenn er sich von einem Zuhörer entfernt oder wegdreht oder sich ein Hindernis zwischen den beiden Spielern befindet. Durch diese Verbindung mit der virtuellen Umgebung sind Spieler in der Lage zu erkennen, wer mit ihnen

spricht, selbst wenn sie von einer unbekannten Stimme angesprochen werden. Ebenso können Umgebungseffekte wie Hall oder Echo und die Position des Sprechers in den Sprachstrom kodiert werden. Diese Möglichkeit besteht für externe Sprachchatanwendungen nicht, da sie keinen Zugriff auf den Kontext der virtuellen Welt haben.

## 4.3 Entwurf der MMOG-Sprachkommunikation

Für den Entwurf einer prototypischen MMOG-Sprachanwendung muss zuerst entschieden werden, welche Architektur verwendet werden soll. Weiterhin muss geplant werden, welche konkreten Gruppenkommunikationsformen wie umgesetzt werden können. Entsprechende Möglichkeiten werden in den nächsten drei Abschnitten diskutiert. Das zugrundliegende Verfahren basiert auf den Arbeiten von T. Plotkowiak [43] und Triebel et al. [60].

### 4.3.1 Architektur

Aus den Anforderungen lässt sich direkt eine geeignete Architektur ableiten. Basierend auf den Skalierbarkeitsanforderungen ergibt sich, dass eine zentrale Verwaltung der Audioströme nicht realisierbar ist. Eine vollständig verteilte Implementierung hingegen ist mit Overlay-Verwaltungskosten verbunden und beeinträchtigt die Effizienz innerhalb der Gruppenkommunikation. Einen Ausweg stellt eine hybride Architektur dar. Diese wird ebenfalls durch Übertragungsprotokolle, wie das Session Initiation Protocol (SIP), unterstützt, die eine Trennung von Signalisierung und Übertragung eines Audiostroms implementieren. D.h., es ist möglich, den zeitkritischen, aber mit nur sehr geringem Bandbreitenbedarf verbundenen Auf- und Abbau von Verbindungen durch einen speziellen Server (Registrar) vorzunehmen und dann die bandbreitenintensive Übertragung der Audioströme direkt zwischen den Clients durchzuführen. Dabei ist diese Architektur unabhängig von der Architektur der virtuellen Welt: Nutzt die virtuelle Welt einen Server, kann dieser auch als Registrar eingesetzt werden; Gibt es keinen Server, muss ein externer Registrar betrieben werden, der ein Interface zur Spiellogik bereitstellt. Die Clients selbst sind dafür verantwortlich, einen persönlichen, kontextbezogenen Audiostrom zu erzeugen und die eingehenden Ströme zu einem einzigen Strom zusammenzufügen. Die-

se Operation kann jedoch mit hohen Kosten verbunden sein und erfordert ein effizientes Vorgehen. Die Form der Gruppenbildung leitet sich ebenfalls aus den Anforderungen ab, jedoch werden diese in abgewandelter Form realisiert. Teamplaygruppen und soziale Gruppen werden indirekt durch lokale und kontextbezogene Gruppen erzeugt.

### 4.3.2 Lokale Gruppenkommunikation

In virtuellen Welten ist das Verlangen von Spielern, miteinander zu kommunizieren, oft mit der Distanz ihrer Avatare in der Spielwelt verbunden. So wäre es beispielsweise hilfreich, wenn zwei Spieler, die Waren auf einem Marktplatz handeln wollen, in einem Gespräch den Preis verhandeln könnten, oder wenn ein Spieler in einer fremden Stadt einen Passanten nach dem Weg fragen könnte.

Dabei lassen sich unterschiedliche Zonen der Kommunikation beobachten, die sich in der realen Welt entwickelt haben und die von den Spielern intuitiv in die virtuellen Welten übertragen werden. Dies ist beispielsweise zu beobachten, wenn befreundete Spieler auf einem öffentlichen (virtuellen) Platz miteinander kommunizieren. Erstaunlicherweise nehmen die Spieler in dieser Situation häufig genau den Abstand ein, den sie auch in der realen Welt nutzen würden. Dabei besteht in der virtuellen Welt kein Anlass dafür, insbesondere wenn über Text kommuniziert wird. Spricht ein Anführer zu einer Gruppe, postiert er sich vor der Gruppe, als ob er vor die Front treten wollte, um zu der Gruppe zu sprechen. Der dabei eingenommene Abstand ist deutlich größer als bei einem privaten Gespräch. Diese Gewohnheiten folgen dem Konzept der *Proxemik*. Das ist ein psychologisches Modell der realen Welt, das von Edward Hall 1966 [21] vorgestellt wurde. Hall erkannte, dass Menschen für verschiedene Kommunikationsformen unterschiedliche Zonen nutzen. Ein Beispiel dafür ist die persönliche Zone, die die nahe Umgebung eines Menschen darstellt. Tritt eine fremde Person in die persönliche Zone, sprich man davon, dass sie jemandem zu nahe kommt, was mit Unbehagen oder einem Empfinden von Bedrohung verbunden ist. Basierend auf Halls Modell können die Eigenschaften der übertragenen Sprachströme anhand der Proximityzonen angepasst werden. Aus dem Modell ergeben sich folgende Zonen:

- *Öffentliche Zone*: Entspricht einer Entfernung in der realen Welt von ca. $5-15\ m$. Spieler in dieser Zone sind zu weit entfernt, um miteinander zu kommunizieren. Es besteht jedoch die Möglichkeit, dass Spieler in naher Zukunft miteinander kommunizieren wollen.

- *Soziale Zone*: Entspricht einer Entfernung von $2-5\ m$. In der sozialen Zone befinden sich die Spieler, die miteinander kommunizieren wollen. Die Kommunikation in dieser Zone ist meist technischer bzw. unpersönlicher Natur.

- *Private Zone*: Entspricht einer Entfernung $< 2\ m$. Die private Zone wird nur von Spielern genutzt, die sich kennen und ein direktes persönliches Gespräch führen wollen.

Die Größe der Zonen ist nicht zwingend konstant. Ebenso wie in der realen Welt können je nach Situation oder Kulturkreis unterschiedliche Radien für das Kommunikationsmodell verwendet werden.

Sobald ein Spieler die öffentliche Zone eines anderen Spielers betritt, wird eine direkte Sprachverbindung zwischen den Clients aufgebaut. Die Signalisierung wird über den Registrar vorgenommen. Die öffentliche Zone dient dabei nur zu Vorbereitung der Verbindung. Sprache wird nicht übertragen. Eine tatsächliche Sprachübertragung beginnt erst, wenn Spieler gegenseitig ihre soziale Zone betreten. Innerhalb dieser Zone wird die Lautstärke des Sprachstroms an die Distanz und die Orientierung der beiden Spieler angepasst. Die Sprache wird in einer niedrigen Qualität mit einer geringen Sampling-Rate aufgenommen. Die niedrige Sampling-Rate wird verwendet, um Bandbreite einzusparen, da davon auszugehen ist, dass innerhalb der sozialen Zone mehrere Spieler gleichzeitig miteinander kommunizieren werden. Betritt ein Spieler die private Zone eines anderen Spielers, wird die Stimme mit voller Lautstärke übertragen, und die Sampling-Rate wird erhöht. Spieler, die diese Zone betreten, wollen in der Regel einen Dialog führen und benötigen dafür die bestmögliche Sprachqualität. Entfernen sich Spieler wieder voneinander, werden Sprachqualität und Lautstärke abgesenkt, bis zuletzt die Verbindung wieder abgebaut wird, wenn die Spieler ihre öffentlichen Zonen verlassen. Durch das Anpassen der Lautstärke und der Bitrate sind die Spieler in der Lage, ihre Kommunikationspartner in der virtuellen Welt zu identifizieren. Besonders durch die Veränderung der

Lautstärke ist der Mensch in der Lage, die Position des Gesprächspartners genauer zu lokalisieren.

### 4.3.3 Kontextbezogene Gruppenkommunikation

Wollen Spieler miteinander kommunizieren, obwohl sie sich nicht in der Nähe befinden, kann ein kontextbezogener Gruppenmodus verwendet werden. So können beispielsweise Freunde, Gilden oder Diskussionsgruppen unabhängig von ihrem aktuellen Standort miteinander kommunizieren. Im Gegensatz zur Zonen-basierten Kommunikation entscheiden die Spieler selbst, welcher Gruppe sie angehören wollen. Allgemein ist die Fluktuation innerhalb dieser Gruppen deutlich geringer als beim ersten Ansatz. Für die Verwaltung der Gruppen wird der Spielserver bzw. ein Registrar eingesetzt, wobei eine verteilte Organisation ebenso möglich ist. In einer reinen P2P-Lösung wäre dafür jedoch ein entsprechendes Protokoll notwendig. Es kann davon ausgegangen werden, dass die Verwaltung von Kommunikationsgruppen keine hohen Anforderungen an einen Server stellt, da lediglich bei Veränderungen der Gruppen Statusnachrichten verschickt werden müssen. Das bedeutet kleine Pakete in geringer Frequenz. Ein leistungsstarker Server ist durchaus in der Lage, die Gruppenverwaltung von mehreren Millionen Spielern zu gewährleisten. Die tatsächlich kritische Menge an Daten, die durch die Audioströme der Sprache erzeugt wird, wird ebenso wie bei dem ersten Ansatz von Client zu Client übertragen. Dabei bleiben die Lautstärke und die Bitrate konstant. Sie werden lediglich an die Größe der Gruppe und an die zur Verfügung stehenden Bandbreiten angepasst.

## 4.4 Implementierung

Unsere Implementierung der prototypischen MMOG-Sprachkommunikation basiert auf der Verwendung einer Bibliothek zur Kodierung der Sprache und eines Netzwerkprotokolls zur Übermittlung der Audioströme. Darüber hinaus wurde die Anwendung in die prototypischen MMOG-Spielanwendung Planet PI4 (siehe Kapitel 3) integriert. Um dabei die Übermittlung von Sprache zu verdeutlichen wurde das Design von Planet PI4 angepasst, indem das Weltraumszenario gegen eine städtische Umgebung ausgetauscht wurde (vgl. Ab-


Abbildung 4.2: Screenshot der prototypischen MMOG-Sprachanwendung. In der Radaransicht (rechts oben) werden Spieler angezeigt, die sich in der öffentlichen, sozialen bzw. privaten Zone befinden

bildung 4.2). Sämtliche Spielefunktionen wie Objektverwaltung, Kollisionserkennung oder Ereignisverwaltung sowie das Netzwerkinterface blieben dabei unverändert. Die Proximityzonen wurden anhand einer 2D-Projektion der 3D-Welt umgesetzt. Eine Besonderheit ergab sich bei der Implementierung der Mischung der Audioströme. Es stellte sich heraus, dass ein naives Mischen von Audioströmen unterschiedlicher Bitraten, wie sie in den unterschiedlichen Proximityzonen verwendet werden, hohe Resamplekosten erzeugt. Um dies zu verhindern, wurde ein zweiphasiges Verfahren implementiert, das ein effizientes Mischen der Audioströme ermöglicht.

### 4.4.1 Sprachkodierung

Für die Implementierung der Sprachkodierung wurde der Speex-Codec verwendet. Speex wurde speziell für *Voice-over-IP (VoIP)*-Anwendungen entwickelt. D.h., die Kodierung wurde für die Übertragung von Sprache unter Verwendung einer geringen Bitrate optimiert. Musik hingegen lässt sich nur unzureichend mit Speex übertragen. Weiterhin wurde in das Design des Codecs einbezogen, dass das Internet als Übertragungsmedium genutzt wird. D.h.,

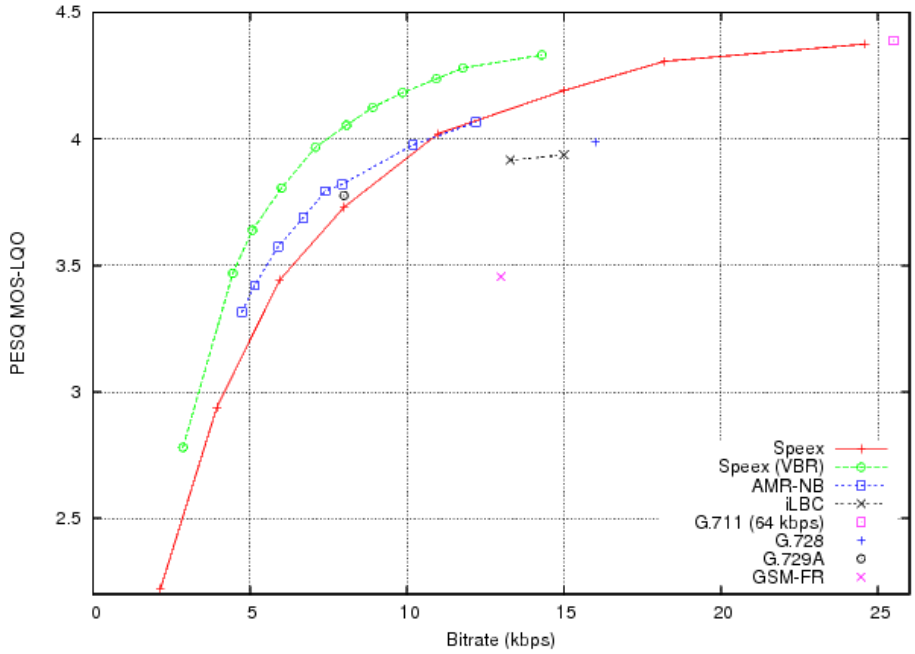



Abbildung 4.3: Übersicht über Bitraten und entsprechende MOS-Werte des Speex-Codecs. Bildquelle: `http://speex.org/comparison/`

Speex ist robust gegenüber Paketverlusten. Paketverstümmelungen, wie sie in Mobilfunknetzen vorkommen können, wurden nicht betrachtet. Ermöglicht wird das durch die Verwendung des *Code Excited Linear Prediction (CELP)*-Kodierungsverfahrens. Speex kann mit drei unterschiedlichen Sampleraten (8,-16,32 kHz) genutzt werden. Zusätzlich können zehn Qualitätsstufen eingestellt werden. Daraus ergeben sich Bitraten von 2-44 kBit/s. Der daraus resultierende MOS ist in Abbildung 4.3 dargestellt. Speex ist eine frei nutzbare Softwarebibliothek[3].

### 4.4.2 Netzwerkprotokoll

Für die Signalisirung und die Übertragung der Audioströme wird eine Kombination des Session Initiation Protocols (SIP)[46, 51] (RFC 2543) und des Realtime Transport Protokols (RTP) [50] verwendet. Die Architektur von SIP definiert dabei einfache Netzwerkelemente, die miteinander interagieren. Obwohl der Aufbau und die Verwaltung von Verbindungen direkt zwischen den

[3] Quelle: `http://en.wikipedia.org/wiki/Speex`

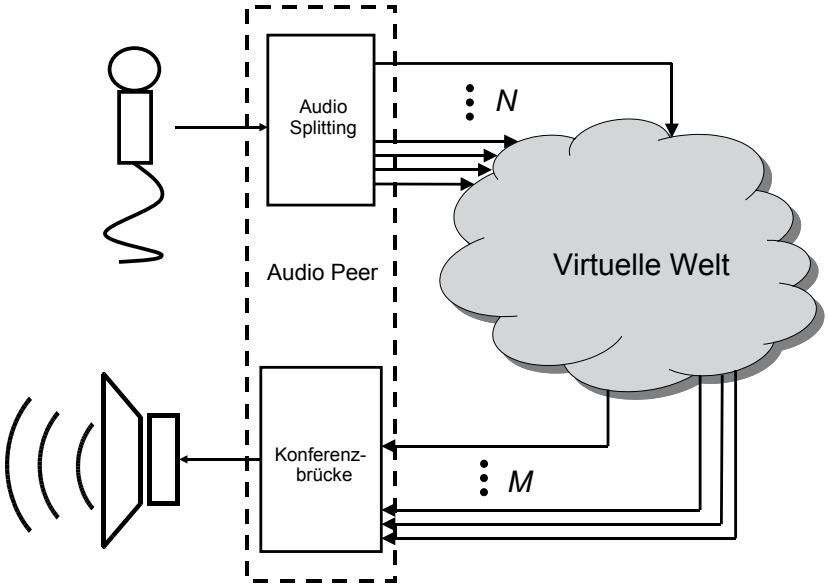



Abbildung 4.4: Die Audio-Peers senden $N$ und empfangen $M$ Ströme. Der persönliche Audiomix wird durch eine lokale Konferenzbrücke erzeugt. Diese muss Audioströme mit unterschiedlichen Samplerate resamplen, um so einen einzigen, gemischten Audiostrom herstellen zu können.

einzelnen Knoten, den SIP User Agents (UAs), stattfindet, können dennoch verschiedene serverbasierte Dienste, wie zum Beispiel die Registrierung von Knoten (Registrar), Weiterleitungen (Redirection Sever) und Routing (Proxy), genutzt werden. Dabei ist die Unterscheidung der Servertypen lediglich logisch. Physikalisch werden alle Dienste durch eine Anwendung implementiert. Weiterhin stellt SIP eine anwendungs- und geräteunabhängige Lösung dar, die alle notwendigen Dienste für die Umsetzung einer Sprachkommunikation anbietet. Im Gegensatz zu reinem RTP und RTCP/SDP bietet SIP ein abstraktes Verbindungsmanagement. Somit ist es nicht notwendig, jede Verbindung einzeln zu verwalten, was gerade bei schnell wechselnden Verbindungen mit erheblichem Aufwand verbunden wäre.

### 4.4.3 Mischen der eingehenden Audioströme

Die lokale Gruppenkommunikation, die in Abschnitt 4.3.2 eingeführt wurde, verwendet variable Sampling-Raten, um die Qualität der Sprache an die Distanz der Spieler anzupassen. Bei einem zuhörenden Spieler resultiert das in einer Vielzahl unterschiedlicher eingehender Audioströme (siehe Abbildung 4.4). Um daraus einen einheitlichen Audiostrom zu erzeugen, müssen die Sampling-Raten der Ströme angepasst werden. Dies ist für jeden einzelnen Strom mit aufwendigen Rechenoperationen verbunden. Ströme, die die gleiche Sampling-Rate nutzen, können hingegen effizient miteinander gemischt werden. Dazu ist lediglich die Addition und Normalisierung der Samples notwendig. Daher wurde für das Mischen der eingehenden Audioströme ein zweiphasiger Prozess verwendet. In der ersten Phase werden alle eingehenden Audioströme anhand der Proximityzonen sortiert. Ströme einer gemeinsamen Zone nutzen dabei stets dieselbe Sampling-Rate. Das Ergebnis ist jeweils ein Strom pro Zone. In der zweiten Phase können dann die verbleibenden Ströme neu gesamplet werden und ergeben schließlich einen einzigen Ausgabestrom. Das sorgt dafür, dass die Kosten für das Mischen der Ströme lediglich linear proportional zu der Anzahl der eingehenden Ströme wächst. Die Anzahl der notwendigen Resampleschritte bleibt dabei konstant. Ebenso kann ein vorhandener Audiostrom des Spiels, wie zum Beispiel Geräusche, hinzugefügt werden.

## 4.5 Experimentelle Ergebnisse

In einem ersten experimentellen Aufbau wurden Funktionsweise und benötigte Bandbreiten der implementierten Sprachkommunikation in einem Zwei-Spieler-Szenario gemessen. Abbildung 4.5 zeigt die benötigte Bandbreite eines Clients in Abhängigkeit von den entsprechendenen Zonen. In diesem Szenario steht ein Spieler still, während sich der zweite bewegt. Der sich bewegende Spieler beginnt in der privaten Zone des anderen und bewegt sich dann über die soziale in die öffentliche Zone. Nach dem Verlassen der öffentlichen Zone wird der Strom vollständig geschlossen. Es wurde angenommen, dass beide Spieler kontinuierlich sprechen. Um dies zu simulieren, wurde ein entsprechender Audiostrom erzeugt, indem ein längerer Text vorgelesen und aufgezeichnet wurde. Der so erzeugte Audiostrom wurde dann simultan für beide Spieler

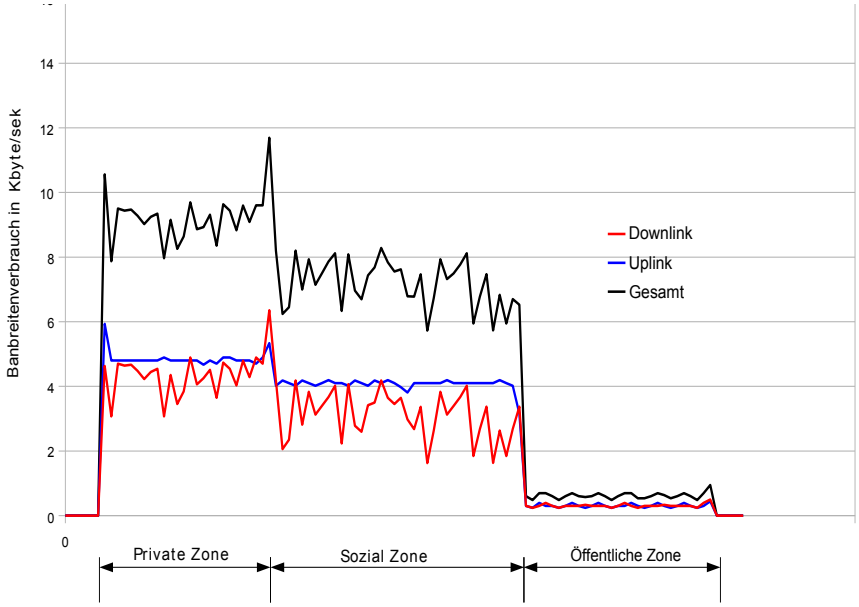



Abbildung 4.5: Bandbreitenbedarf eines Clients in Abhängigkeit von der Distanz der Gesprächspartner.

abgespielt. Der verwendete Speex-Codec nutzt dabei ein Stimmabtastintervall von 20 ms. Das entspricht einer Paketrate von 50 Datenpaketen pro Sekunde. In der privaten Zone wurde eine Sampling-Rate von 16 $kHz$ verwendet, was einer Bitrate von $16,8$ $kBit/s$ ($\approx$ 2 $kB/s$) entspricht. In der sozialen Zone wurde die Sampling-Rate dann auf $8kHz$ reduziert, woraus eine Bitrate von 11 $kBit/s$ ($\approx 1,5$ $kB/s$) resultierte. Die Grafik zeigt, dass der Bandbreitenbedarf deutlich über der Codec-Bitrate liegt. Das ist bedingt durch den Overhead der Netzwerkheader. Insgesamt werden 50 Pakete pro Sekunde mit jeweils 60 Header-Bytes gesendet. Das resultiert in einem Overhead, der durch das Netzwerk verursacht wurde, von 3 $kB/s$. Gemittelt ergab die Messung einen Gesamtverbrauch (Sende- und Empfangsdaten) von 9 $kB/s$ in der privaten Zone, 7 $kB/s$ in der sozialen Zone und weniger als 1 $kB/s$ in der öffentlichen Zone. Die große Varianz der Downlink-Bitrate ist durch die unterschiedliche Hardware der beiden Clients begründet. In einem Fall wurde dabei ein Mikrofon verwendet, das zu starkem Rauschen neigte. Dadurch kam die Speex-Silence Surpression quasi nicht zum Einsatz. Der zweite Client konnte dagegen selbst kurze Sprechpausen erkennen und dadurch Daten einsparen.

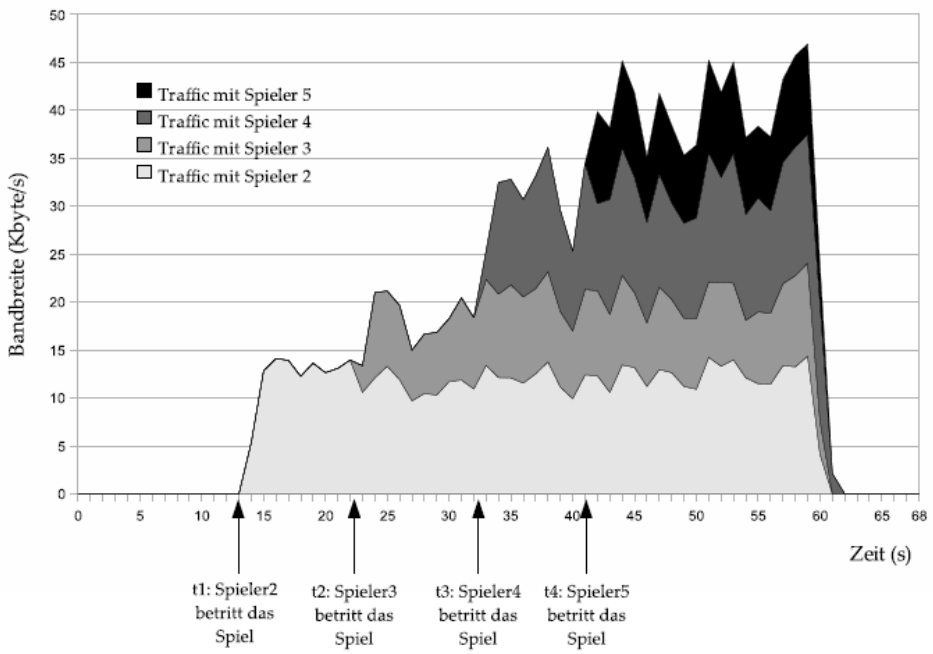



Abbildung 4.6: Bandbreitenbedarf eines Clients in einer Kontextgruppe mit fünf Teilnehmern.

In einer zweiten Messung wurde der gesamte Bandbreitenverbrauch einer Kontextgruppe mit fünf Teilnehmern untersucht. Dabei wurden keine Proximityzonen verwendet, und alle Clients nutzten eine Sampling-Rate von 16 $kHz$. Abbildung 4.6 zeigt den Bandbreitenverbrauch der einzelnen Teilnehmer, die der virtuellen Welt im Abstand von zehn Sekunden beigetreten sind. Gemessen wurde das gesamte Datenaufkommen (Sende- und Empfangsweg). Es ist zu erkennen, dass pro Spieler ein Gesamtdatenaufkommen von ca. 10 $kB/s$ entsteht. Bei einer Gruppengröße von 25 Teilnehmern entsteht somit ein Gesamtvolumen von ca. 250 $kB/s$.

## 4.6 Fazit

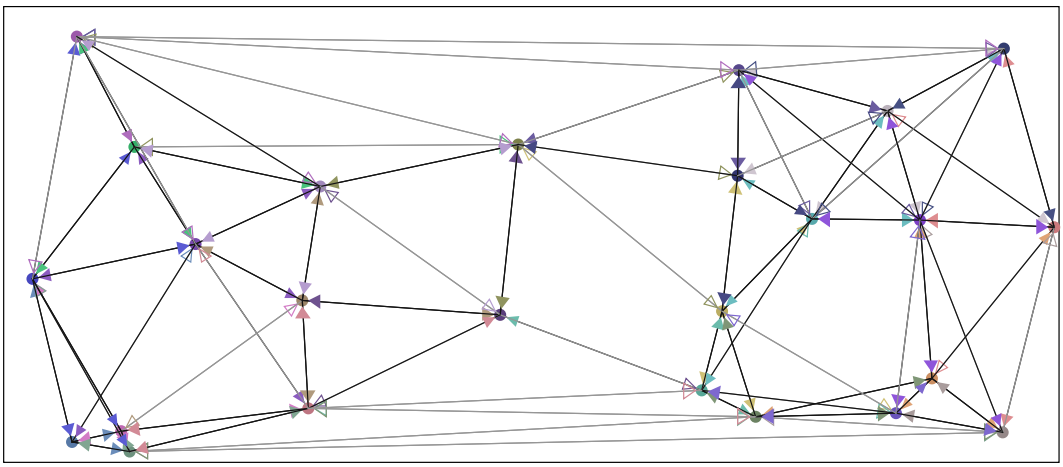
In diesem Kapitel wurde eine prototypische Sprachkommunikationsanwendung für MMOGs vorgestellt. Diese basiert auf einer hybriden Architektur, die einerseits eine zentrale Signalisierung der Audioverbindungen nutzt, andererseits die ressourcenkritischen Audioströme direkt zwischen Clients überträgt. Zusätz-

lich wurde das Realwelt-Modell der Proxemik integriert, um lokale Gruppen in Abhängigkeit von definierten Kommunikationszonen automatisch zu generieren. Aufgrund der Integration in die prototypische Spielanwendung Planet PI4 war es zudem möglich, die Sprachkommunikation mit der Spiellogik zu verbinden und somit die Sprache in ihre virtuelle Umgebung einzubetten. Dadurch konnten Distanz und Orientierung der Sprecher und der Einfluss der Umgebung der Sprache als zusätzlicher Effekt hinzugefügt werden.

Erste experimentelle Ergebnisse haben gezeigt, dass das Design der Anwendung den Anforderungen gerecht wird. Die gesamte Leistungsfähigkeit des Systems (z.B. Qualität oder Stabilität bei großen Teilnehmerzahlen) ist jedoch noch offen, da große Simulationen oder Feldversuche nicht möglich waren. Dennoch hat die Anwendung in dem getesteten Rahmen gut funktioniert und gezeigt, dass das System durch die Verwendung von SIP leicht erweitert werden kann. So sind beispielsweise Verbindungen in das Telefonnetz möglich. Festnetz- oder Mobilfunkteilnehmer können dadurch problemlos den Kommunikationsgruppen hinzugefügt werden.

# Kapitel 5

# P2P-Overlay



## 5.1 Problemstellung

Die Übertragung der Sprache ist ein wichtiger Netzwerkdienst für MMOGs. Ihr Einsatz ist jedoch erst dann sinnvoll, wenn bereits eine grundlegende Kommunikationsinfrastruktur existiert. Diese Infrastruktur besteht aus dem Austausch von Status-, Update- und Systemnachrichten. So wird aus dem statischen Modell einer virtuellen Umgebung eine lebendige Welt. Spieler können miteinander kommunizieren und interagieren. Sie können gemeinsam Aufgaben lösen, Handel treiben oder einfach nur verfolgen, was andere Spieler tun.

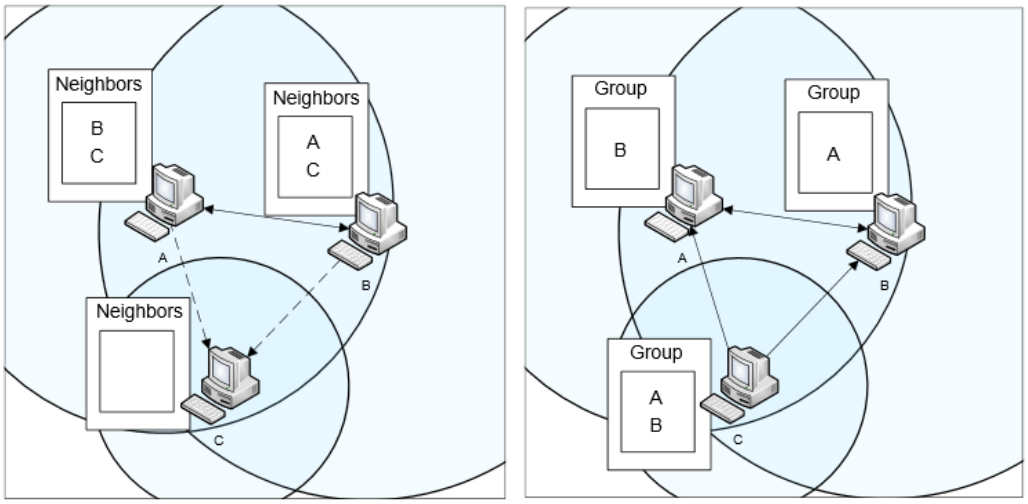
Der intuitive Ansatz für die Umsetzung eines derartigen Netzwerkes ist, wie bereits erwähnt, die Nutzung einer Client-/Server-Architektur. Dabei übernimmt ein dedizierter Server die gesamte Steuerung und Koordination der

Welt. Alle Spieler nehmen eine Clientrolle ein und senden die spiele-relevanten Informationen an den Server. Dieser verwaltet den Gesamtzustand der Welt und informiert in regelmäßigen Abständen alle Clients über Veränderungen. Aufgrund der zentralen Verwaltung wird bei diesem Ansatz jedoch die Teilnehmerzahl der virtuellen Welt durch die Leistungsfähigkeit des Servers und die damit verbundenen Kosten limitiert. Daher ist das Ziel dieses Kapitels die Entwicklung eines alternativen Netzwerkansatzes, der in der Lage ist, eine unbegrenzte virtuelle Welt zu schaffen, die kostenfrei operiert. Dazu wird eine Netzwerkarchitektur vorgestellt, die einen verteilten Publish-/Subscribe-Mechanismus für die Verbreitung der Updatenachrichten nutzt. Darüber hinaus werden weitere Komponenten umgesetzt, die zum Betreiben einer derart verteilten virtuellen Welt notwendig sind. Abschließend wird die Leistungsfähigkeit des Ansatzes anhand verschiedener Simulationen untersucht.

## 5.2 Architektur

Das Ziel der hier vorgestellten Architektur ist es, ein Netzwerk zu erzeugen, das kostenfrei operiert und dennoch in der Lage ist, den Anforderungen eines MMOGs gerecht zu werden (Mildner et al. [41, 40]). Dabei stehen in diesem konkreten Szenario eine unbegrenzte Teilnehmerzahl, ein stark dynamisches Spielerverhalten und eine möglichst hohe Echtzeitfähigkeit im Vordergrund. Aufgrund der hohen Dynamik wurden dabei keine Overlaystrukturen, wie zum Beispiel DHTs, Voronoidiagramme oder Superpeers verwendet. Vielmehr kommt ein vollständig verteiltes P2P-Netzwerk als Basis für den Austausch von Nachrichten zum Einsatz. In einem derartigen Netzwerk sind die Knoten selbst für das Management der Informationen verantwortlich. Das bedeutet, dass Knoten entweder erkennen müssen, welche anderen Knoten über ihre lokalen Informationen benachrichtigt werden müssen, oder sie müssen in der Lage sein herauszufinden, wer für sie relevante Informationen besitzt.

Aktuelle Systeme nutzen meist den ersten Ansatz. Dabei kann es jedoch zu einem erheblichen Kommunikationsoverhead kommen. Insbesondere, wenn Spieler nicht das selbe AOI haben, können sogar Inkonsistenzen entstehen. Wie in Abbildung 5.1a zu erkennen ist, können zwei Probleme bei einem senderorientierten Ansatz entstehen: Einerseits erhalten Knoten Nachrichten, die für sie




(a) Senderorientiertes Modell (b) Publish-/Subscribe-Modell

Abbildung 5.1: Beim senderbasierten Modell sendet jeder Teilnehmer Nachrichten an die Teilnehmer in seiner AOI. Beim Publish/Subscribe-Modell schreibt sich jeder Teilnehmer in die Gruppen der Teilnehmer ein, die in seiner AOI liegen.

nicht relevant sind. Das erhöht die Datenrate, ohne Nutzen zu erzeugen. Darüber hinaus bekommen Spieler möglicherweise nicht alle Informationen über ihr AOI, da das Interesse an Information nicht bidirektional ist, wenn Spieler keine gemeinsame AOI haben. Das kann im schlimmsten Fall zu Netzwerkinkoinsistenzen führen. Um dieses Problem des senderorientierten Modells zu umgehen, kann ein *Publish/Subscribe-System* verwendet werden, um die Informationen zu verbreiten. Anstatt die Informationen an Spieler in der eigenen AOI zu versenden, bekommen Nutzer die Informationen, die ihren Ursprung in der AOI haben, indem sie sich bei allen Knoten in ihrer AOI anmelden. Zusätzlich publiziert jeder Knoten alle Informationen an die entsprechenden Gruppen, an die sich andere Knoten *subscriben* können. Somit kontrollieren die Knoten nicht die Gruppen, die Nachrichten empfangen, sondern sie stellen die Gruppe der Knoten zusammen, an deren Informationen sie Interesse haben. Mit einer derart unstrukturierten, publish-/subscribe-basierten P2P-Architektur kann einerseits ein effizientes dynamisches Overlay implementiert werden, andererseits ist eine optimale Abdeckung der Informationen, die für einen Spieler relevant sind, möglich (siehe Abbildung 5.1b).

## 5.3 Netzwerkdienste und -mechanismen

Nachdem die Entscheidung für die grundlegende Netzwerkarchitektur getroffen wurde, müssen die notwendigen Netzwerkdienste und Mechanismen umgesetzt werden. Dabei gilt es, zunächst die Basiskomponenten IM und GED zu erstellen. Damit verbunden ist ein Mechanismus, der die Konsistenz des Overlays sicherstellt. Dabei auftretende Fehler müssen automatisch behoben werden können. Das Ergebnis dieser zusammenwirkenden Dienste und Mechanismen ist eine robuste, echtzeitfähige Basisstruktur einer verteilten virtuellen Welt. Diese Struktur wird zusätzlich um einen Geocast-Dienst für die virtuelle Welt erweitert, der es ermöglicht, Informationen gezielt an relevante Regionen zu senden.

### 5.3.1 Interest Management - Game Event Dissemination

Interest Management und Game Event Dissemination sind zwei Netzwerkdienste, die je nach Architektur sehr eng miteinander verbunden sein können. In vielen Ansätzen stellt die AOI eine direkte Verbindung der beiden Dienste dar, da anhand dieses Gebietes festgelegt wird, zu welchen Knoten Verbindungen aufgebaut werden und welche Informationen für einen Knoten relevant sind. Dabei wird meist eine kreisförmige AOI, die sich um den Avatar des Spielers erstreckt, verwendet. Dieser Ansatz funktioniert problemlos in wenig bevölkerten Gebieten. Steigt die Dichte jedoch an, kann es zu einem hohen Bandbreitebedarf kommen, zum Beispiel, wenn sich eine große Zahl von Spielern an einer Stelle versammelt *(Density Problem)*. Je nach Anzahl der Spieler kann dabei die zur Verfügung stehende Bandbreite eines Knotens überschritten werden. Da derartige Szenarien häufig in MMOGs vorkommen, muss das Netzwerk in der Lage sein, damit umzugehen. Eine Möglichkeit dafür ist es, eine dynamische AOI zu verwenden [3]. Überschreitet die Zahl der Spieler in der AOI eine Obergrenze, wird die AOI verkleinert, so dass eine bestimmte Obergrenze an Verbindungen nicht überschritten wird. Dadurch verringert sich jedoch auch die Sicht und die Handlungsweite eines Avatars. Verringert sich die Dichte, kann der Radius der AOI wieder vergrößert werden. Eine zweite Möglichkeit besteht darin, in einer Verdichtungssituation die Updatefrequenz zu erniedrigen. Diese Technik kommt in den Verfahren Donnybrook und Colys-

seus zum Einsatz [7, 8]. Da diese Möglichkeit jedoch die Antwortzeiten erhöht, ist sie für zeitkritische Spiele nicht uneingeschränkt einsetzbar.

Der hier vorgestellte Ansatz basiert dagegen nicht auf einer dynamischen AOI, sondern auf einer festen Obergrenze von Verbindungen. So werden stets höchstens $n$ Verbindungen zu den wichtigsten Nachbarn aufgebaut, wobei $n$ ein Parameter ist, der anhand verschiedener Faktoren festgelegt werden kann. Beispielsweise kann er abhängig von der zur Verfügung stehenden Bandbreite festgelegt werden. Dabei muss nicht immer die maximale Anzahl der Verbindungen aufgebaut werden. Befindet sich beispielsweise kein anderer Spieler in der Sicht oder Interaktionsreichweite, kann die Anzahl der Verbindungen auf ein Minimum reduziert werden. Ungeklärt ist dabei zunächst noch, wie die jeweiligen Nachbarn ermittelt, die Konsistenz des Overlays gewährleistet und die genutzten Verbindungen verwaltet werden können.

### Erkennung der Nachbarn

Da es keine zentralen Server oder Super-Peers gibt, müssen die Knoten selbst miteinander kommunizieren, um wechselnde Nachbarn in der virtuellen Welt zu erkennen. Dafür gibt es zwei klassische Ansätze:

- Knoten tauschen periodisch ihre Nachbarschaftlisten aus.
- Dedizierte Knoten (Watchdogs) sind dafür verantwortlich, die Umgebung für andere Knoten zu überwachen und sie über Veränderungen zu informieren.

Dabei benötigt der erste Ansatz keine zusätzlichen Watchdogs, jedoch produziert er Overhead, wenn sich die Nachbarschaftsverhältnisse nicht ändern. In dem hier vorgestellten Ansatz werden dynamische Watchdogs verwendet, die einer Kombination der beiden Ansätze entsprechen.

Nutzer verwalten keine Gruppe von Watchdogs, sondern senden periodisch (mit einer geringen Updatefrequenz) Anfragen an alle AOI-Nachbarn, die eine Liste der eigenen Nachbarn enthalten. Kennt ein Nachbar einen Knoten, der für den anfragenden Knoten geeigneter ist, überträgt er dessen Identität und Position. Der anfragende Knoten kann dann selbst entscheiden, ob er eine Verbindung zu diesem Knoten aufbauen will oder nicht. Dieses Vorgehen

hat den Vorteil, dass die Knoten keine Statusinformationen über Nachbarn anderer Knoten verwalten müssen. Zusätzlich können Knoten lokal eine Nachbarschaftserkennung initiieren. Das folgt dem Prinzip, dass jeder Knoten seine Verantwortung selbst wahrnimmt.

### Konsistenz des Overlays

In einer gleichverteilten virtuellen Welt ist ein rein nachbarbasierter Ansatz ausreichend, um die Spieler mit einem konsistenten Overlaynetzwerk zu verbinden. Spieler in MMOGs neigen jedoch dazu, sich an bestimmten Orten zu sammeln oder sich in Gruppen zu bewegen. So kann der Nachbaransatz zu Overlayinkonsistenzen [1] und sogar zu Partitionen der virtuellen Welt führen, wenn die Distanz zwischen Gruppen von Spielern größer als der Abstand der jeweiligen nächsten Nachbarn ist. Da eine Partitionierung nicht automatisch behoben werden kann, muss sie um jeden Preis vermieden werden.

Um diese Konsistenz zu gewährleisten, muss der nachbarschaftsbasierte Ansatz um eine zusätzliche Art von Verbindungen *(NetConnectors)* erweitert werden. Diese Verbindungen nutzen die Struktur der virtuellen Welt, um die Konsistenz des Overlays zu erhalten. Basierend auf dem Prinzip, dass jeder Nutzer mindestens eine Verbindung in jede virtuelle Richtung betreiben soll, kann eine Menge von konsistenzerhaltenden Verbindungen definiert werden. Diese Verbindungen stimmen nicht notwendigerweise mit den nächsten Nachbarn überein. Daher ist die Menge der NetConnectors immer ein Tradeoff zwischen den Interessen der Nutzer und der Konsistenz des Netzwerks. pSense [49] nutzt einen ähnlichen Ansatz, jedoch sind die dabei verwendeten Sensorknoten immer außerhalb der AOI, wodurch stets eine zusätzliche Last erzeugt wird. Um die Übereinstimmung der beiden Mengen zu maximieren, sind NetConnectors immer die nächsten Knoten in dem jeweiligen Gebiet. Allgemein werden NetConnectors nach den folgenden Regeln ausgewählt:

- Gegeben sei eine $D$-dimensionale Welt, dann muss die Anzahl $N$ der NetConnectors $N \geq 2^D$ erfüllen.

[1] In diesem Abschnitt wird dabei der Begriff Konsistenz ausschließlich für die Overlaykonsistenz verwendet. Konsistenz der Spielezustände ist ein zusätzlicher Mechanismus, der unabhängig von zugrundeliegenden Verbindungsalgorithmen ist, daher wird dieser hier nicht betrachtet.

- Der Raum um einen Avatar wird in $N$ gleiche Zonen unterteilt.
- In jeder der resultierenden Zonen muss eine Verbindung zu dem nächsten Nachbarn aufgebaut werden.

Im Falle einer 2-dimensionalen Welt[2] kann eine Einteilung in die minimale Anzahl von vier Regionen dadurch erreicht werden, dass die Welt an den Koordinatenachsen mit dem Ursprung an der Position des Avatars geteilt wird. Während eine minimale Anzahl von NetConnectors die Konsistenz des Netzwerks sicherstellen soll, kann eine zusätzliche Anzahl dafür genutzt werden, die Stabilität des Netzes auch bei Verbindungsfehlern aufrechtzuerhalten.

### Verwaltung der Verbindungen

Da das IM und die Konsistenz des Overlays auf korrekten Verbindungen basieren, müssen diese akkurat verwaltet werden. Dabei können Konstellationen auftreten, in denen benachbarte Spieler eine unterschiedliche AOI haben. Das kann dazu führen, dass Verbindungen nur unidirektional aufgebaut werden und ein Ungleichgewicht im Spielerlebnis entsteht. So können Spieler, wenn sie sich nur einseitig sehen, einen unfairen Vorteil daraus ziehen, indem sie gegnerische Spieler bereits bekämpfen, bevor diese sie sehen können.

Um das zu verhindern, sollten Verbindungen immer bidirektional sein, selbst wenn Spieler nicht explizit an einer Verbindung interessiert sind. Diese sekundären Verbindungen werden nur verwendet, solange der Partner daran interessiert ist. Ist der primäre Partner nicht mehr interessiert, wird die Verbindung abgebaut. Um die unterschiedlichen Zustände von Verbindungen zu verwalten, kann eine Tabelle mit Verbindungszuständen genutzt werden. Konkret können dabei folgende Zustände vorkommen:

**Verbindungsphase:** Bestimmt den allgemeinen Zustand der Verbindung (verbindend, trennend, verbunden)

**Zustand der NetConnectors:** Drückt aus, ob entweder der Partnerknoten oder der Knoten selbst ein NetConnector ist.

[2]Viele NVE nutzen eine 2 bzw. zwei einhalb dimensionale Repräsentation der Welt, In diesen Fällen ist die Nutzung einer zweidimensionalen Projektion der Welt für das Erstellen des Overlays sinnvoll.

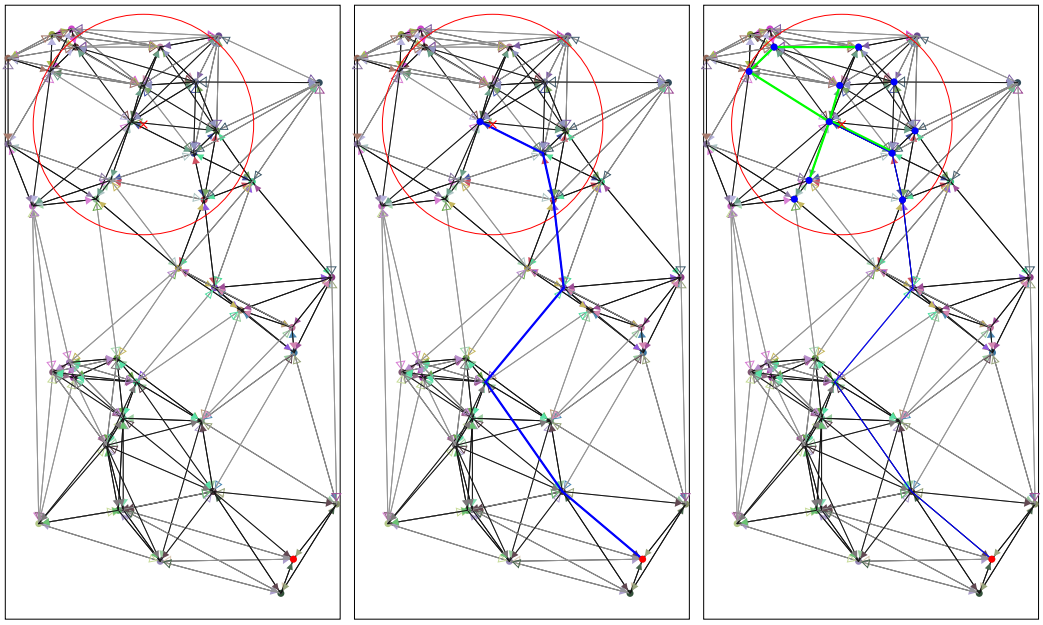


Abbildung 5.2: Vom Ausgangsknoten (roter Knoten) wird der Geocast zunächst ins Zielgebiet weitergeleitet (blau). Nach der Aktivierung durch den Knoten, der dem Zielpunkt am nächsten liegt, wird er im Zielgebiet verteilt (grün).

**Art der Verbindung:** Besagt, ob die Verbindung eine primäre oder sekundäre Verbindung ist.

Da jeder der Werte für eingehende und ausgehende Verbindungen separat gespeichert werden muss, entsteht eine Tabelle, die aus sechs Variablen pro Eintrag besteht. Solange nur ein Partner an einer Verbindung interessiert ist, hat er die volle Kontrolle über die Verbindung. Haben beide Partner ein Interesse an der Verbindung, müssen sie einen Verantwortlichen aushandel. Ein Problem entsteht, wenn beide Verbindungen sekundär sind. In diesem Fall wäre ein rein lokales Verbindungsmanagement nicht ausreichend, da kein Knoten die Verbindung beenden könnte. Um diese Situation zu umgehen, senden die Knoten periodisch mit einer niedrigen Frequenz Statusnachrichten, um die Verbindungszustände zu synchronisieren. So wird die Verwaltung der Zustände robuster.

### 5.3.2 Geocast

Bei normalen Operationen werden lediglich Nachrichten zwischen Nachbarn ausgetauscht. Das ist zwar ausreichend für Spieleupdates oder Interaktionen zwischen Nachbarn, jedoch kann es auch vorkommen, das Nachrichten an Spieler geschickt werden müssen, die sich außerhalb der AOI befinden. Ein Beispiel dafür wäre eine Waffe mit einer Gebietswirkung und einer großen Reichweite, wie etwa bei einer Rakete. Um auch derartige Spielmechanismen unterstützen zu können, wurde ein Geocast-Algorithmus entwickelt, der es erlaubt, Nachrichten an beliebige Regionen der virtuellen Welt zu schicken. Die Schlüsseleigenschaft dieser Nachrichten ist, dass sie nicht direkt an Spieler adressiert werden, sondern an eine entsprechende Region, die durch die Koordinaten des Mittelpunktes und eine Beschreibung der Form definiert wird. Der Algorithmus besteht dabei aus zwei Schritten: Erstens wird eine Nachricht vom Sender zur Zielregion weitergeleitet, ohne dabei ausgewertet zu werden. Zweitens muss der Inhalt an die Spieler, die sich innerhalb der Region befinden, verbreitet werden. Diese können dann den Inhalt auswerten. Während die Weiterleitung in die Zielregion durch einfache Unicastverbindungen realisiert werden kann, ist das Verteilen der Information innerhalb der Region schwieriger. Eine triviale Lösung wäre es, das Zielgebiet mit Nachrichten zu fluten, um alle Teilnehmer zu erreichen. Dies ist jedoch mit einer hohen Netzwerklast verbunden. Um diese Last zu minimieren, können NetConnectors eingesetzt werden, um einen Geocast im Zielgebiet zu verbreiten. NetConnectors sind dafür besonders geeignet, da sie geografisch um einen Knoten verteilt sind.

Im Detail sieht der Geocast-Algorithmus wie folgt aus: Ein Geocast besteht aus einer Nachricht, die einen Zielpunkt, einen Verteilradius (in diesem Fall kreisförmig), eine eindeutige Identifzierungsnummer und den Inhalt der Nachricht enthält. Beginnend mit dem Sender schickt jeder beteiligte Knoten die Nachricht an denjenigen bekannten Knoten, der den geringsten Abstand zum Zielpunkt hat. Gibt es keinen weiteren Knoten mehr, und die Nachricht befindet sich bereits im Zielgebiet, endet die erste Phase. In der zweiten Phase wird der Geocast im Zielgebiet verteilt, indem er vom Zentrum bis zum Rand ausgebreitet wird. Jeder Knoten verschickt die Nachricht an seine NetConnectors, wenn die folgenden Bedingungen erfüllt sind:

- Der NetConnector muss sich innerhalb des Ausbreitungsgebiets des Geocasts befinden.
- Der NetConnector muss weiter vom Zielort entfernt sein als der Knoten, von dem die Nachricht empfangen wurde.
- Der eigene Knoten darf nicht mehrfacher Empfänger der Nachricht sein.

Durch diesen Mechanismus kann der Anstieg der durch das Flooding verschickten Nachrichten linear gehalten werden. Endloses Flooding wird dadurch vermieden, dass sich die Nachrichten vom Ziel wegbewegen, bis sie die Grenze der Region erreicht haben. Um ein mehrfaches Empfangen von Nachrichten zu verhindern müssen eingehende Nachrichten eine gewisse Zeit gecached werden.

## 5.4 Evaluation

Um einen Überblick über die Leistungsfähigkeit der entwickelten Konzepte zu erlangen, wurde das Netz mit verschiedenen Parametern getestet. Dafür wurde eine eigens entwickelte Simulationsumgebung eingesetzt. Diese stellt eine kontrollierte Umgebung dar, die in der Lage ist, unterschiedliche Ereignisse und Szenarien anhand verschiedener Simulationsparameter zu erzeugen. Im Folgenden werden der konkrete Simulationsaufbau und die dabei verwendeten Parameter vorgestellt.

### 5.4.1 Simulationsaufbau

Für die Evaluation des Overlays wurden zwei Simulationsmodi verwendet. Zum Einen wurden für *statische* Analysen Bewegungen und periodisch gesendete Positionsnachrichten deaktiviert. Dieser reduzierte Funktionsumfang wurde eingesetzt, um das Verhalten bei großen Teilnehmerzahlen zu simulieren. Zum Anderen berücksichtigen *dynamische* Simulationen sämtliche Aktionen der Knoten und schließen insbesondere Bewegungen und Schüsse der Spieler mit ein. Sie wurden eingesetzt, um die Konsistenz und Stabilität des Netzwerkes unter dynamischen Bedingungen zu simulieren. Aufgrund der aufwändigen Bewegungssimulationen sind in diesem Modus jedoch nur Simulationen mit bis zu 500 Knoten möglich.

Die Simulation wurde auf einem leistungsfähigen Rechner ausgeführt, dem neben vier Prozessoren 8 GB Arbeitsspeicher zur Verfügung standen. Die Ressourcen konnten von der Simulationsumgebung voll genutzt werden, da ein Parallelisierungsmechanismus in die Simulation integriert wurde.

### 5.4.2 Simulationsparameter

Die Leistungsfähigkeit des Netzwerkes wird von unterschiedlichen Kenngrößen beeinflusst. Um die jeweiligen Auswirkungen analysieren zu können, wurden diese Kenngrößen in der Simulationsumgebung parametrisiert eingesetzt. D.h., durch die Variation der Parameter kann das Verhalten des Netzwerkes unter unterschiedlichen Bedingungen untersucht werden. Dabei kamen folgende Parameter zum Einsatz:

**Anzahl der Knoten:** Um die Skalierbarkeit des Netzes abschätzen zu können, kann die Anzahl der Knoten im Netz variiert werden. Unter ansonsten konstanten Bedingungen kann so die Belastung auf Netz und Teilnehmer untersucht werden. Verhält sich diese konstant, erfüllt das Overlay die Skalierbarkeitsanforderungen.

**Anzahl der Nachbarn / NetConnectors:** Grundsätzlich können die Werte für Nachbarn und NetConnectors unabhängig voneinander gesetzt werden. Dennoch stehen sie in einem gewissen Zusammenhang. Durch Erhöhung der Anzahl der Nachbarn oder NetConnectors verbessert sich jeweils die Stabilität des Netzwerkes auf Kosten einer Mehrbelastung der Teilnehmer. Der Zuwachs der Stabilität verhält sich jedoch weder linear noch unabhängig voneinander. D.h., es muss ein geeigneter Kompromiss zwischen Belastung und Stabilität gefunden werden, der ein ausgewogenes Verhältnis von Nachbarn und NetConnectors beinhaltet.

Neben den reinen Zahlenwerten gibt es weitere Größen, durch die das Netzwerk beeinflusst werden. Dazu zählt im Besonderen das Bewegungsmuster der Teilnehmer. Vor allem Muster, bei denen sich viele Teilnehmer auf engem Raum aufhalten, stellen ein Problem dar. Daher ist es sinnvoll, unterschiedliche Bewegungsmuster für die Simulationen auszuwählen.

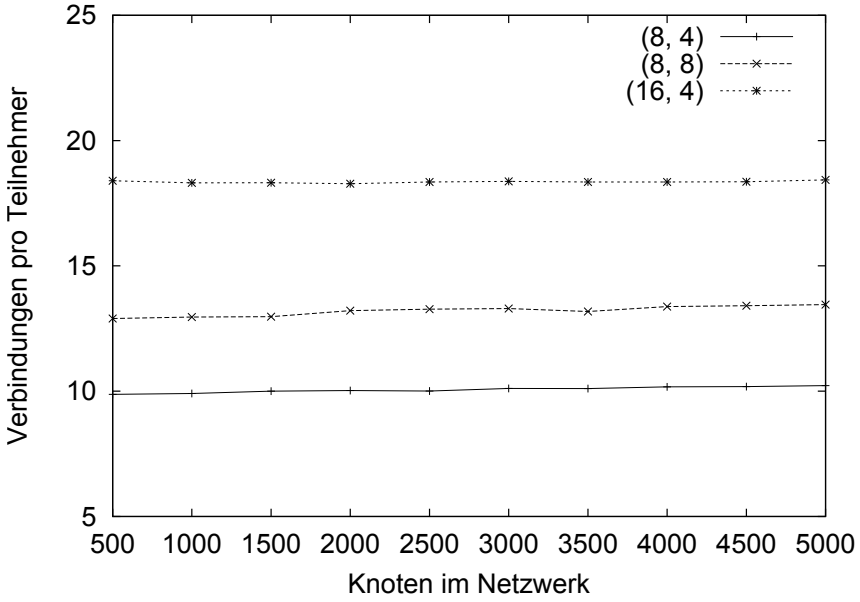



Abbildung 5.3: Die Anzahl der Verbindungen pro Knoten bliebt bei steigender Knotenanzahl und verschiedenen Konfigurationen annähernd konstant. Die Konfigurationen sind im Format (Nächste Nachbarn, NetConnectors) für die Kombinationen (8,4), (8,8) und (8,16) abgebildet

### 5.4.3 Statische und dynamische Simulationen

Einen ersten Anhaltspunkt für die Skalierbarkeit des Netzes bietet die relative Belastung der Knoten bei steigender Knotenanzahl. Hierfür wurde das Netz mit steigender Knotenanzahl simuliert und dabei jeweils die Verbindungen pro Knoten gemessen. Dabei wurde mit Knotenzahlen zwischen 500 und 5000 gearbeitet. Die Simulation wurde mit drei Parameterkonfigurationen ausgeführt, bei denen jeweils die Anzahl der nächsten Nachbarn und der NetConnectors geändert wurde. Gemessen wurde jeweils die Summe der ein- und ausgehenden Verbindungen eines Knotens im voll aufgebauten Netz. Die Auswertung zeigt, dass die Anzahl der Verbindungen nahezu konstant bleibt (siehe Abbildung 5.3). D.h., dass durch das Hinzufügen von Knoten nur geringe Abweichungen in der Anzahl der Verbindungen entstehen.

#### Einfluss von nächsten8Nachbarn auf die Anzahl der Verbindungen

Ein wichtiger Faktor, der über die Anzahl der Verbindungen entscheidet, ist die Anzahl der nächsten Nachbarn. Um diesen Zusammenhang zu untersu-

chen, wurden Simulationen mit unterschiedlicher Anzahl von nächsten Nachbarn durchgeführt. Die Anzahl der NetConnectors wurde dabei auf den Minimalwert von vier gesenkt, um die Messungen möglichst wenig zu beeinflussen. Die Simulationen wurden jeweils mit 1000 Knoten im Netz ausgeführt. Bei der Messung wurden folgende fünf Metriken angewandt:

**Verbindungen:** Hierbei wird die Gesamtzahl an Verbindungen eines Teilnehmers festgehalten.

**Ausgehende NetConnectors:** Hierbei wird die Anzahl der Knoten gezählt, die den jeweiligen Teilnehmer als NetConnector ausgewählt haben.

**Ausgehende Nachbarn:** Analog zu der Zahl der ausgehenden NetConnectors wird die Zahl der ausgehenden Nachbarverbindungen festgehalten. Da die beiden Mengen sich überschneiden können, kann die Summe höher liegen als die Zahl der Gesamtverbindungen.

**Reine eingehende NetConnectors:** Dies ist die Anzahl der Verbindungen, die ein Teilnehmer zu NetConnectors aufgebaut hat, die selbst keine Nachbarn des Teilnehmers sind. Dieser Wert sollte so gering wie möglich sein.

**Unerwünschte Verbindungen:** Zuletzt wird die Anzahl einseitig ausgehender Verbindungen festgehalten. Die Verbindungspartner sind weder Nachbar noch NetConnector des jeweiligen Teilnehmers, und die Verbindungen werden nur vom Partner kontrolliert. Auch hier ist ein geringer Wert besser, da diese Verbindungen Teilnehmer belasten, ohne ihnen einen direkten Nutzen zu bringen.

Bei zunehmender Anzahl der Nachbarn ist eine lineare Steigerung der Verbindungen zu beobachten (siehe Abbildung 5.4). Gleichzeitig nimmt die Menge der reinen NetConnector-Verbindungen ab, so dass die vorhandenen Verbindungen mehr mit dem Interessenbereich des Teilnehmers übereinstimmen. Zudem ist folgendes Verhältnis der Werte zu beobachten: Die Summe aus nächsten Nachbarn, reinen NetConnectors und unerwünschten Verbindungen ergibt die Gesamtzahl der Verbindungen eines Teilnehmers. So bleibt der Abstand von der Anzahl nächster Nachbarn zur Gesamtzahl der Verbindungen mit steigender Nachbaranzahl nahezu konstant.

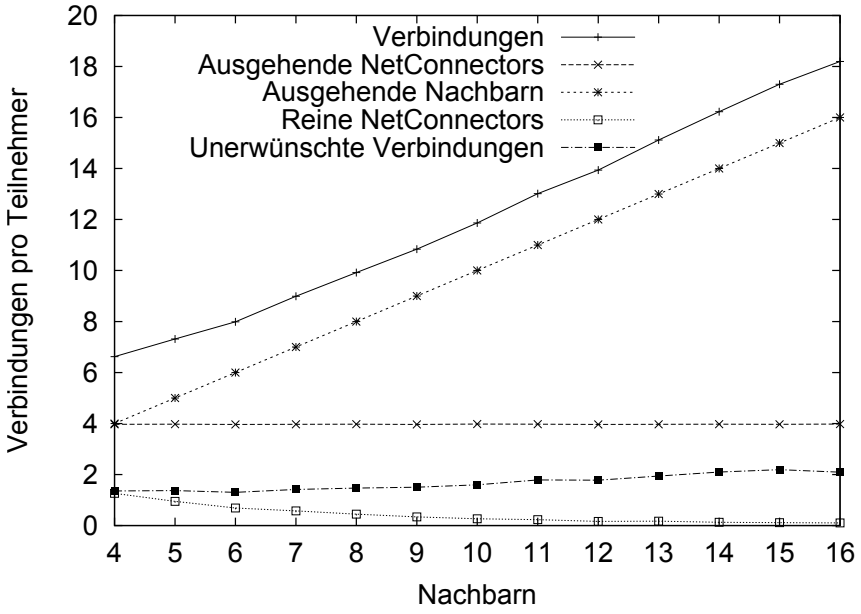



Abbildung 5.4: Bei der Erhöhung der Nachbarverbindungen nimmt die Anzahl der Verbindungen linear zu, während das Verhältnis reiner NetConnector-Verbindungen abnimmt.

### Einfluss von NetConnectors auf die Anzahl der Verbindungen

Analog zur vorigen Messung wurden Simulationen mit unterschiedlicher Anzahl von NetConnectors ausgeführt. Dabei wurde überprüft, welchen Einfluss diese auf die Anzahl der Verbindungen haben. Die Anzahl der nächsten Nachbarn wurde auf vier fixiert. Auch hier ist ein lineares Wachstum der Verbindungen zu beobachten (siehe Abbildung 5.5). Im Gegensatz zu der Entwicklung bei den nächsten Nachbarn steigt jedoch der Abstand der Anzahl von NetConnectors zur Gesamtanzahl der Verbindungen. Den größten Einfluss haben dabei unerwünschte Verbindungen, da zwischen den Teilnehmern mehr Verbindungen zu ansonsten nicht benötigten NetConnectors aufgebaut werden müssen.

### Netzwerklast von Geocast-Nachrichten

Der Geocast wurde so konzipiert, dass er starken Gebrauch von NetConnectors zur Verteilung der Nachricht macht. In diesem Zusammenhang wurden zwei Simulationsdurchläufe durchgeführt. Beim ersten Lauf wurde die Anzahl der vom Geocast erreichten Knoten zwischen 500 und 2500 variiert, um Aufschlüs-

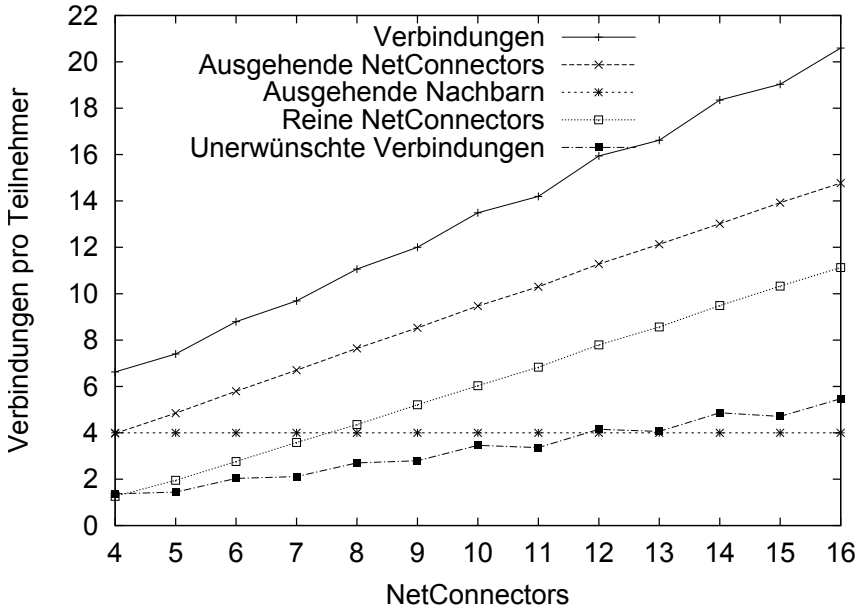
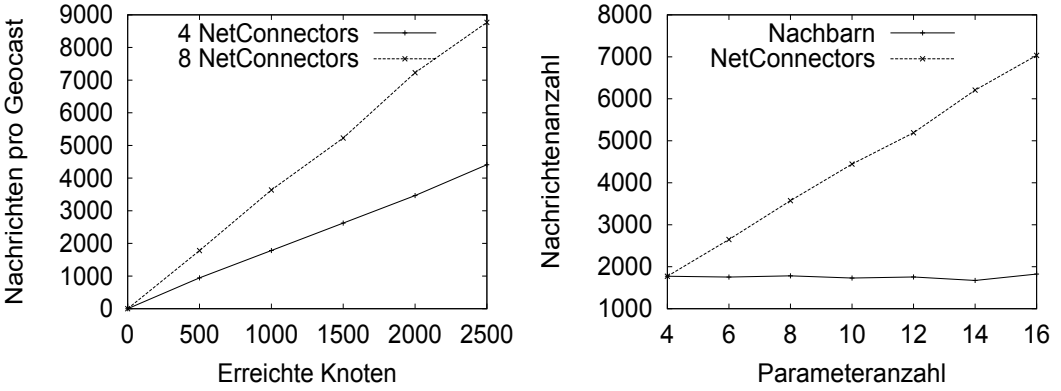



Abbildung 5.5: Durch die Erhöhung der NetConnector-Verbindungen nimmt auch die Anzahl unerwünschter Verbindungen zu.

se über das Nachrichtenwachstum zu erlangen. In der zweiten Phase wurde mit fixierten 1000 Knoten die Anzahl der nächsten Nachbarn und NetConnectors variiert. Die Auswertung des ersten Laufs zeigt, dass sich die Anzahl der nötigen Nachrichten bei steigender Knotenzahl linear verhält (siehe Abbildung 5.6a). Dies ist eine positive Eigenschaft, da gerade bei Flooding-Mechanismen die Gefahr einer hohen Netzwerklast besteht. In dem zweiten Lauf zeigte sich, dass die Anzahl der Nachrichten alleine von der Anzahl der NetConnectors abhängt (siehe Abbildung 5.6b). Dabei verhält sich die Anzahl der NetConnectors proportional zu den benötigten Nachrichten eines Geocast. Dies ist damit zu erklären, dass Geocast-Nachrichten ausschließlich an NetConnectors weitergeleitet werden.

### Einfluss von nächsten Nachbarn und NetConnectors auf die Stabilität des Netzes

Um Erkenntnisse über die Stabilität des Netzes zu erlangen, wurden die Reparaturfähigkeiten des Netzes getestet. Dazu wurden 20% bis 80% der Knoten aus einem konsistenten Netz entfernt. Nach dem durch die restlichen Knoten eingeleiteten Reparaturvorgang wurde überprüft, ob sich das Netz wieder in




(a) Entwicklung der Netzwerklast von Geocast-Nachrichten

(b) Einfluss von NetConnectors auf Geocast-Nachrichten

Abbildung 5.6: Die Netzwerklast von Geocast-Nachrichten verhält sich linear. Sie wird alleine von der Anzahl der NetConnectors beeinflusst.

einem konsistenten Zustand befindet. Dabei ist zu beachten, dass schon ein Knoten, der die Verbindung zu den anderen verloren hat, einen irreparablen Zustand verursacht. Das Netz ist in diesem Fall nicht mehr konsistent. Um allerdings einen Vergleich ziehen zu können, wurde jeweils der Anteil der Knoten, die die Verbindung zum Hauptteil der Knoten verloren haben, gemessen. Auf diese Weise konnte die Schwere der Konsistenzverletzungen beziffert werden. Wenn das Netz jedoch in fünf oder mehr Partitionen geteilt wurde, wurde es im Gesamten als inkonsistent markiert und die Konsistenz somit auf 0% festgelegt. Die Simulationen wurden jeweils mit 250 Knoten ausgeführt. Die Knoten hatten dabei eine statische Positionen. Zu jeder Konfiguration wurden fünf Durchläufe gestartet, aus denen anschließend der Durchschnittswert gebildet wurde. Um möglichst vergleichbare Ergebnisse zu produzieren, kam bei jedem Durchlauf der identische Knotenaufbau zum Einsatz.

**Einfluss nächster Nachbarn:** Um den Einfluss der nächsten Nachbarn analysieren zu können, wurde die Anzahl der NetConnectors auf vier festgelegt, wobei die Anzahl der nächsten Nachbarn stufenweise von vier auf 16 erhöht wurde. Die Auswertung zeigt einen Anstieg der Stabilität bei steigender Nachbaranzahl (siehe Abbildung 5.7). Während bei vier Nachbarn schon bei 30% Verlust erste Konsistenzverletzungen auftreten, kann die Stabilität des Netzes mit 16 Nachbarn bei bis zu 60% Verlust gewährleistet werden. Der Reparaturvorgang kann alle Knoten, zwischen denen

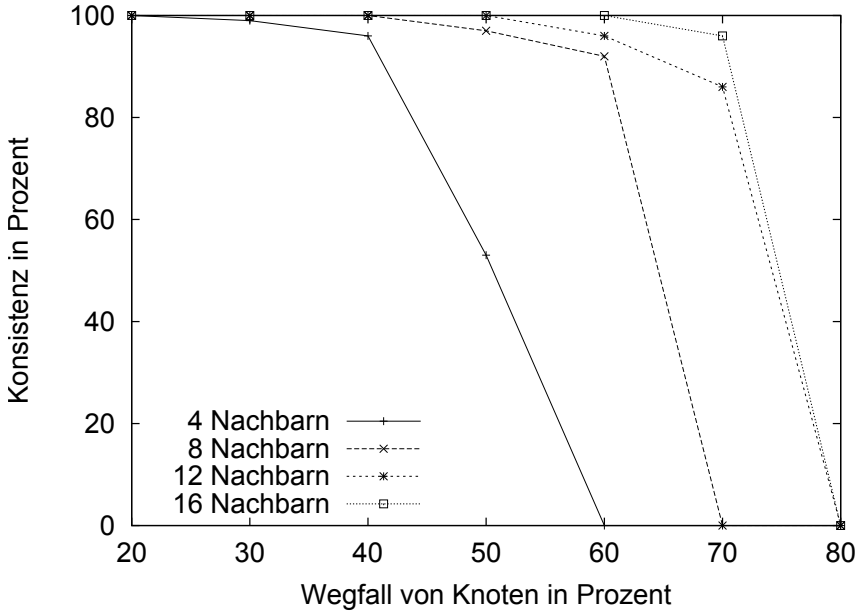



Abbildung 5.7: Bei der Erhöhung der Nachbarverbindungen steigt die Reparaturfähigkeit des Netzes

eine Verbindung besteht, in einen konsistenten Zustand überführen.

**Einfluss NetConnectors:** Entsprechend der ersten Simulation wurde im zweiten Aufbau die Anzahl der der nächsten Nachbarn auf vier festgelegt und die Anzahl der NetConnectors von vier auf 16 erhöht. Bei der Auswertung zeigt sich ein ähnliches Bild wie bei der ersten Simulation (siehe Abbildung 5.8). Auch hier bringt die Erhöhung der Anzahl der NetConnectors eine gesteigerte Stabilität. Dies geschieht allerdings in einem höheren Maße als bei den nächsten Nachbarn. So kann bei 16 NetConnectors selbst beim Wegfall von 80% der Knoten noch eine Konsistenz von über 90% erreicht werden.

### Einfluss von Churn auf die Stabilität des Netzes

Neben dem Wegfall von Knoten kann auch ein ständiger Wechsel der Teilnehmermenge zu Inkonsistenzen führen. Dazu wurden Simulationen durchgeführt, bei denen Teilnehmer zum Netz hinzugefügt wurden und nach einer gewissen Zeit die Verbindung wieder selbstständig beendeten. Im Rahmen der Tests

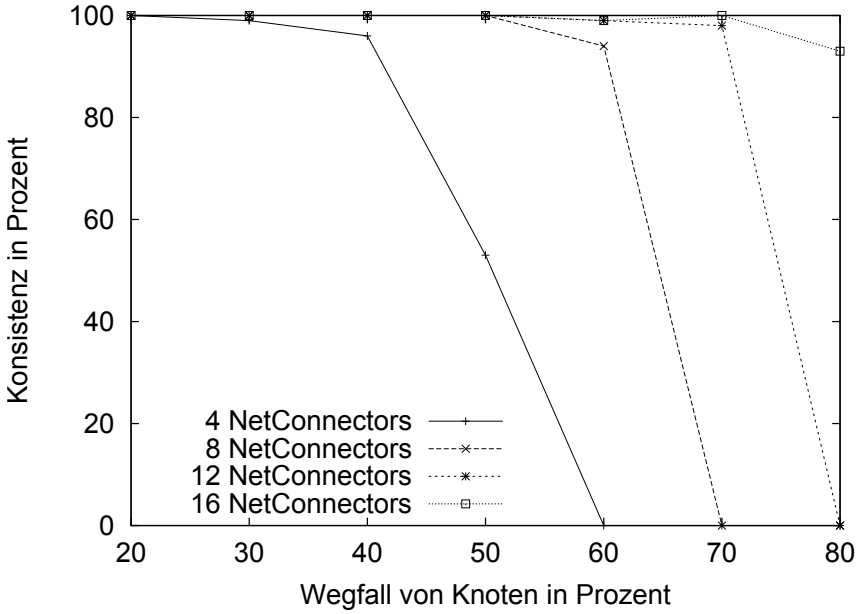



Abbildung 5.8: Durch die Erhöhung der NetConnector-Verbindungen kann die Konsistenz bei Knotenverlusten von bis zu 70% gewährleistet werden

kam es zu keinen Inkonsistenzen, die durch solche Teilnehmerwechsel verursacht wurden. Im Vergleich zu den beiden vorigen Durchläufen erfolgten die Netzaustritte hier beabsichtigt und konnten entsprechend von den verbleibenden Knoten behandelt werden. Somit verblieb das Netz nach Austritten in einem konsistenten Zustand und konnte neue Knoten ohne Probleme aufnehmen.

### Einfluss der Bewegungsstrategie auf die Anzahl der Verbindungen

Alle zuvor ausgeführten Tests wurden ohne Bewegung der Knoten ausgeführt. Zwar lieferten diese Simulationen wichtige Ergebnisse über grundlegende Eigenschaften des Netzes, jedoch können nur mit Bewegung praxisnahe Ergebnisse generiert werden. Zunächst wurde deshalb in Simulationen der Einfluss der Bewegung auf die Anzahl der Verbindungen pro Teilnehmer untersucht. Für die Generierung der Ergebnisse wurden Simulationen mit bis zu 200 Knoten ausgeführt. Dabei kamen folgende Bewegungsmuster zum Einsatz:

**Zufall:** Der Teilnehmer wählt einen zufälligen Punkt in seiner Umgebung, auf den er zusteuert.

**Verfolgung:** Der Teilnehmer wählt einen Nachbarknoten aus, dem er daraufhin für eine gewisse Zeit folgt.

**Gruppen:** Mehrere Teilnehmer finden sich auf der Karte zu einer Gruppe zusammen und ziehen daraufhin in der Gruppe mittels zufällig gewählter Koordinaten über die Karte.

**Versammlung:** Alle Teilnehmer sammeln sich um einen Punkt der Karte und bewegen sich anschließend zufällig in einem Bereich um diesen Punkt.

**Zufall und Gruppen:** Je die Hälfte der Knoten bewegt sich in Gruppen bzw. zufällig über die Karte.

**Zufall und Verfolgung:** Dieses Muster ersetzt das Grundmuster „Verfolgung“, da der alleinige Einsatz dieses Musters zu einer Ansammlung aller Knoten in einem Punkt führt. Die beiden Strategien werden gleichmäßig auf die Knoten verteilt.

**Gemischt:** In diesem Muster sind sowohl Zufalls-, Gruppen- als auch Verfolgungsmuster enthalten. 20% der Knoten bewegen sich dabei zufällig, der Rest teilt sich gleichmäßig unter den beiden anderen Mustern auf.

Zur Ermittlung der Werte wurde jede Simulation für eine festgesetzte Zeit von drei Minuten ausgeführt. Nach Ablauf dieser Zeit wurde die Simulation pausiert und die momentanen Verbindungsinformationen gespeichert. Dies diente dazu, dass sich alle Knoten zunächst ihrem Muster entsprechend positionieren konnten, bevor die Ergebnisse ermittelt wurden. Bei der Auswertung der Ergebnisse sind einige Anomalien zu erkennen, die besonders bei Mustern mit Versammlungsstrategie auftreten. Da diese Muster eine derart hohe Last erzeugten, dass eine fehlerfreie Ausführung der Simulation bei großen Knotenzahlen nicht mehr möglich war, sind die Ergebnisse nicht repräsentativ. Trotzdem lässt sich eine Tendenz erkennen. Die niedrigsten Werte werden mit dem zufälligen Muster erreicht (siehe Abbildung 5.9).

Dies kann damit erklärt werden, dass hierbei alle Knoten möglichst gleichmäßig über die Karte verteilt sind. Bei Ansammlungen von Knoten hingegen steigt die Anzahl der Verbindungen deutlich an.

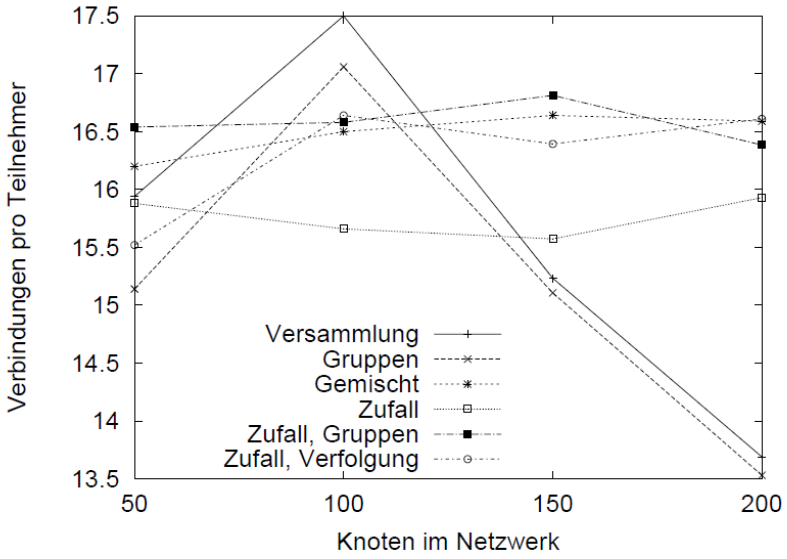



Abbildung 5.9: Beim Einsatz verschiedener Bewegungsmuster ändert sich die durchschnittliche Anzahl der Verbindungen. Gruppen- und Versammlungsmuster erzeugen ab 100 Teilnehmern eine solch hohe Last, dass keine korrekten Ergebnisse mehr produziert werden können

### Einfluss der Bewegungsstrategie auf die Verbindungsdynamik

Im Zusammenhang mit der Belastung der Teilnehmer bei unterschiedlichen Bewegungsstrategien ist nicht nur die reine Anzahl von Verbindungen interessant, sondern auch die Dynamik der Verbindungen. Denn durch häufiges Auf- und Abbauen von Verbindungen entsteht bei den Teilnehmern eine nicht unerhebliche Last. Zudem wirken sich kurzlebige Verbindungen negativ auf das Nutzerempfinden aus. Um die verschiedenen Bewegungsmuster vergleichen zu können, wurde folgende Metrik gewählt: Über einen festen Zeitraum von drei Minuten wurde die Anzahl der Verbindungsaufbauten pro Teilnehmer ermittelt. Ein niedriger Wert bedeutet eine geringe Belastung und eine niedrige Frequenz der Teilnehmerwechsel.

Auch hier tauchen bei den Ergebnissen die im vorigen Lauf erkannten Anomalien auf. Trotzdem können Aussagen über die ermittelten Werte gemacht werden. In dieser Simulation zeigt sich noch stärker die große Belastung bei großen Ansammlungen von Knoten (siehe Abbildung 5.10).

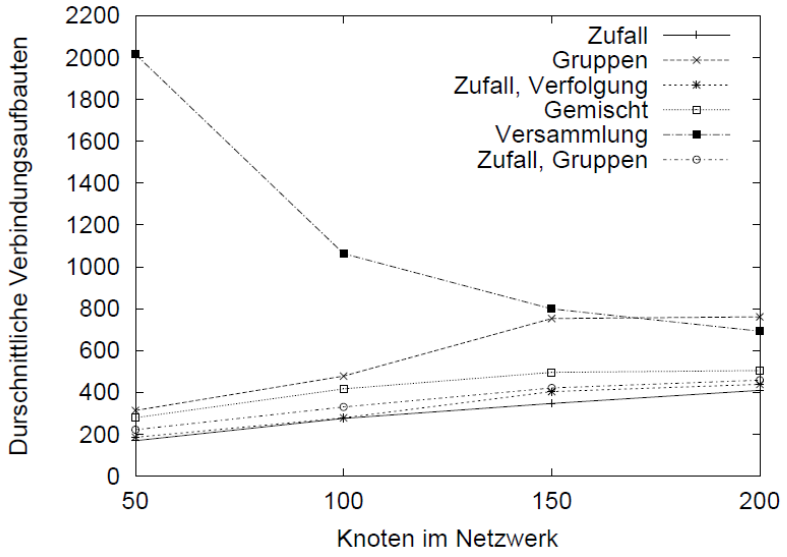



Abbildung 5.10: Die Bewegungsmuster haben einen großen Einfluss auf die Dynamik der Verbindungen. Das Versammlungsmuster erzeugte wiederum aufgrund der hohen Last keine korrekten Ergebnisse

## 5.5 Fazit

In diesem Kapitel wurde ein Konzept für ein verteiltes Overlay vorgestellt, das für den Einsatz in einer virtuellen Welt entworfen wurde. Das Konzept basiert einerseits auf einem geografischen Publish-/Subscribe-Mechanismus zum Aufbau von lokalen Verbindungen, andererseits auf dynamischen Watchdogs, den NetConnectors, die zur Stabilisierung des Netzwerkes eingesetzt werden. Zusätzlich wurde ein geografischer Broadcastmechanismus implementiert. Anhand einer speziell entwickelten Simulationsumgebung wurden sämtliche Funktionen des Overlays getestet und ausgewertet. Die Ergebnisse geben dabei einen ersten Eindruck von der Leistungsfähigkeit des Overlays. So ist zu erkennen, wie sich die Anzahl der lokalen Verbindungen bei zunehmender Teilnehmerzahl und bei verschiedenen Bewegungsmustern verhält. Ebenso lässt sich die Stabilität des Netzwerkes unter verschiedenen Bedingungen abschätzen. Dadurch hat sich gezeigt, dass die NetConnectors einen höheren Einfluss auf die Stabilität des Netzes haben als die direkten Nachbarn. Zusätzlich hat sich in realen Testreihen mit der prototypischen Spieleanwendung Planet PI4 gezeigt, dass das Overlay für kleine Spielergruppen zuverlässig funktioniert und Kom-

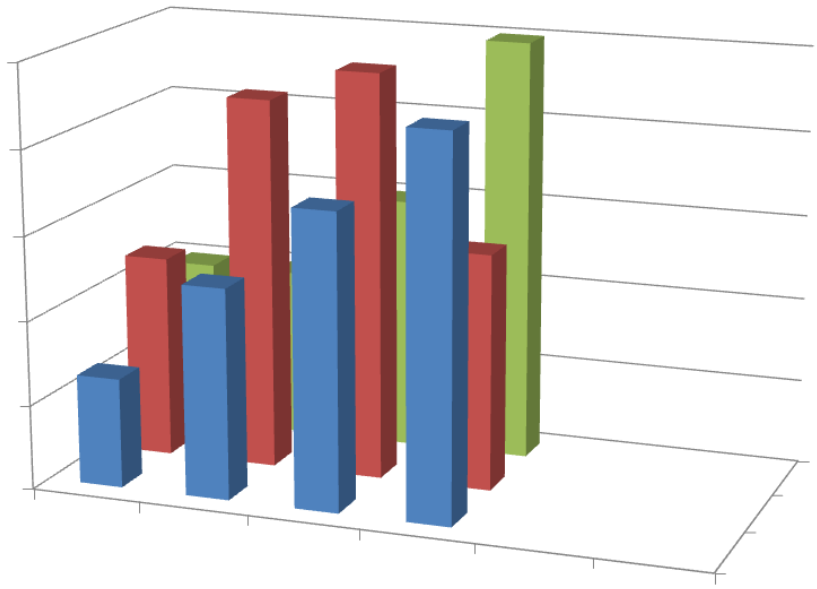
munikationspartner stets verzögerungsfrei erkannt wurden.

Dennoch lässt sich weder anhand der Simulationen, noch anhand der realen Evaluationen eine scharfe Aussage über die Leistungsfähigkeit des Overlays treffen. In den Simulationen ist zwar erkennbar, dass sich die Anzahl der Verbindungen je nach Bewegungsmuster stark verändert und im Extremfall der globalen Versammlung sogar zu fehlerhaften Ergebnissen führt, aber welches Bewegungsmuster einer realen Spielesession entspricht, ist dabei noch ungeklärt (siehe Abschnitt 6.7. Diese Information ist jedoch für eine aussagekräftige Bewertung notwendig.

In den realen Testreihen waren zwar perfekte Bewegungsmuster der Spieler vorhanden, jedoch war die Teilnehmerzahl zu klein. D.h., es ist nicht klar, wie sich das Overlay unter großer Last verhält. Darüber hinaus wurden sämtliche Ergebnisse isoliert betrachtet. Es lässt sich also nicht abschätzen, ob das Overlay im Vergleich zu anderen (VON, pSense, etc.) leistungsfähiger ist oder nicht. Ein derartiger Vergleich erfordert eine einheitliche Testumgebung, standardisierte Metriken und eine realistische und reproduzierbare Last, anhand derer die Overlays gleichmässig evaluiert werden können. Die Erstellung eines derartigen *Benchmarks* ist Inhalt des folgenden Kapitels.

# Kapitel 6

# Benchmarking



## 6.1 Problemstellung

Ziel dieses Kapitels ist es, grundsätzlich zu klären, wie Netzwerkarchitekturen für MMOGs auf ihre Leistungsfähigkeit überprüft und miteinander verglichen werden können. Dazu werden zunächst grundlegende Überlegungen und Anforderungen eines derartigen Benchmarks zusammengestellt. Als nächstes wird die Methodik, anhand derer der Benchmark entworfen wird, festgelegt. Im Rest des Kapitels werden dann einzelne Komponenten wie Metriken, Benchmarkumgebung und Lastgenerierung vorgestellt. Zuvor erfolgt jedoch eine Festlegung

von zentralen Begriffen, die im Zusammenhang mit Benchmarking verwendet werden.

## 6.2 Terminologie

**Szenario:** Ein Benchmarkszenario definiert das Anwendungsgebiet eines Benchmarks. Es wird durch die Beschreibung aller Funktionalitäten, die ein System erfüllen muss, festgelegt. Konkret wird dazu ein Interface benutzt. In verteilte Systeme beinhaltet das in der Regel komplexe Strukturen, bei denen einzelne Komponenten miteinander wechselwirken. Daher erfordert gerade die Festlegung des Szenarios oft ein tiefgreifendes Verständnis aller involvierten Funktionalitäten.

**System under Test (SUT):** Das SUT bezeichnet eine Komponente bzw. eine Anwendung, die ein Szenario umsetzt. In Fall eines MMOG-Neztzwerkbenchmarks besteht ein SUT aus der Implementierung eines Overlaynetzwerkes.

**Last:** Die Arbeitslast bzw. Netzwerklast eines verteilten Systems bezeichnet die Menge der Nachrichten, die zwischen einzelnen Knoten ausgetauscht werden muss, um den Inhalt einer Anwendung umzusetzten. Bei einer File-Sharing-Anwendung sind das beispielsweise Suchanfragen nach Daten wie Musikstücken oder Filmen, die zwischen den Nutzern ausgetauscht werden. Steuerungsbefehle, die für die Verwaltung des Systems selbst genutzt werden, zählen nicht als Arbeitslast. Da die Arbeislast durch Operationen der Nutzer entsteht, müssen diese bei einem Benchmark simuliert werden, um eine entsprechende Last zu erzeugen. Dabei unterscheidet man synthetisch und real erzegten Lasten[1]. Bei synthetischer Lastgenerierung werden Nachrichten basierend auf Anwendungsmodellen erzeugt, bei realer Lastgenerierung wird dafür eine bestehende Anwendung verwendet. Um alle Bereiche eines SUT abdecken zu können, wird die Last parametrisiert. Für Benchmarks von verteilten Systemen gibt es dafür zwei gängige Dimensionen (siehe [15]): horizontal und ver-

[1]Die Unterscheidung bezieht sich dabei auf die Art und Weise, wie die Last simuliert wird.

tikal. Horizontale Skalierung bedeutet eine Veränderung der Teilnehmerzahl, vertikale Skalierung verändert die Last, die die jeweiligen Nutzer erzeugen. Die Benchmarkingkomponente, die für die Erzeugung der Last verantwortlich ist, wird als Lastgenerator bezeichnet.

**Metriken:** Um die Qualität eines Systems numerisch ausdrücken zu können, werden Funktionen verwendet, die in der Lage sind, einzelne Leistungen zu messen und auf einen Qualitätswert abzubilden. Das gesamte Leistungsprofil ergibt sich dann aus einer sinnvollen Kombination der einzelnen Qualitätswerte. Gängige Metriken in verteilten Systemen sind beispielsweise durchschnittliche Antwortzeit, Datendurchsatz oder maximale Fehlerrate.

Allgemein wird dabei zwischen Mikro- und Makrometriken unterschieden. Diese Unterscheidung bezieht sich auf den Kontext des zu beurteilenden Dienstes. Soll eine globale Funktion untersucht werden, kommen Makrometriken zum Einsatz; ist dagegen das Ziel, die Wirkungsweise eines internen Prozesses zu ermitteln, werden Mikrometriken verwendet. So wäre beispielsweise bei einer verteilten Datenbank die Antwortzeit auf eine Datenbankanfrage eine Makrometrik, dagegen die Vollständigkeit der Routingtabellen eine mögliche Mikrometrik. Mikrometriken liefern meist Informationen über die Ursache der erzielten Leistungen (z.B., wenn die Antwortzeit einer verteilten Datenbank aufgrund unvollständiger Routingtabellen hoch ist). Allerdings können sie oft nicht für einen Leistungsvergleich verwendet werden, da unterschiedliche Systeme auch unterschiedliche innere Strukturen haben können. Somit besteht beim Entwurf eines Benchmarks ein wichtiger Schritt darin, eine möglichst aussagekräftige, aber dennoch allgemeingültige Menge an Metriken zu wählen. Die Benchmarkingkomponente, in der die Metriken implementiert sind, heißt Monitor.

**Test-Prozeduren:** Allgemein gibt es zwei unterschiedliche Testverfahren, die für Benchmarks verwendet werden können: statische Tests und Variationstests. Statische Tests verwenden eine festgelegte Arbeitslast. Die Leistung des Systems wird direkt durch die Metriken gemessen. Im Gegensatz dazu erhöhen Variationstests dynamisch die Last, mit der das

System belegt wird. Zusätzlich werden QoS-Grenzen definiert. Das Ergebnis des Benchmarks besteht dann aus der maximalen Last, unter der das System die geforderten QoS-Grenzwerte noch einhalten konnte.

## 6.3 Anforderungen

Ziel eines Benchmarks ist es, die Leistungsfähigkeit eines Systems zu analysieren und zu bewerten. Das geschieht auf der Basis von Qualitätsaspekten, die im Vorfeld festgelegt werden müssen. Um ein aussagekräftiges Ergebnis liefern zu können, muss ein Benchmark grundlegende Anforderungen erfüllen (Definition nach Jain [28]):

**Reproduzierbarkeit:** Die Ergebnisse eines Benchmarks müssen stets nachvollziehbar sein. D.h., es muss gewährleistet sein, dass bei der Wiederholung eines Benchmarks zu einer anderen Zeit, an einem anderen Ort und mit einere anderen Implementierung das gleiche Ergebnis erzielt wird.

**Vollständigkeit:** Alle relevanten Komponenten eines Systems müssen von einem Benchmark abgedeckt werden. Um Vollständigkeit zu gewährleisten, ist eine umfassende Kenntnis des Anwendungsszenarios notwendig. Werden relevante Teile ausgelassen, hat das einen direkten Einfluss auf die Ergebnisse.

**Skalierbarkeit:** Ein Benchmark darf nicht auf eine bestimmte Skalierung beschränkt sein. Gerade bei verteilten Systemen entstehen viele Anforderungen erst ab einer kritischen Anzahl von Nutzern. Daher muss beim Entwurf eines Benchmarks stets darauf geachtet werden, dass ein System in jeder relevanten Skalierung getestet werden kann.

**Authentizität:** Ein Benchmark muss immer auf einer realistischen Arbeitslast basieren. Die Arbeitslast hat den größten Einfluss auf das Ergebnis eines Benchmarks. Somit hängt auch die Qualität eines Benchmarks von der Art der Lastgenerierung ab. Die Lastgenerierung ist der zentrale Aspekt bei der Entwicklung eines Benchmarks.

**Neutralität:** Ein Benchmark darf nicht auf ein bestimmtes Produkt zugeschnitten sein. D.h., alle Komponenten des Benchmarks müssen derart

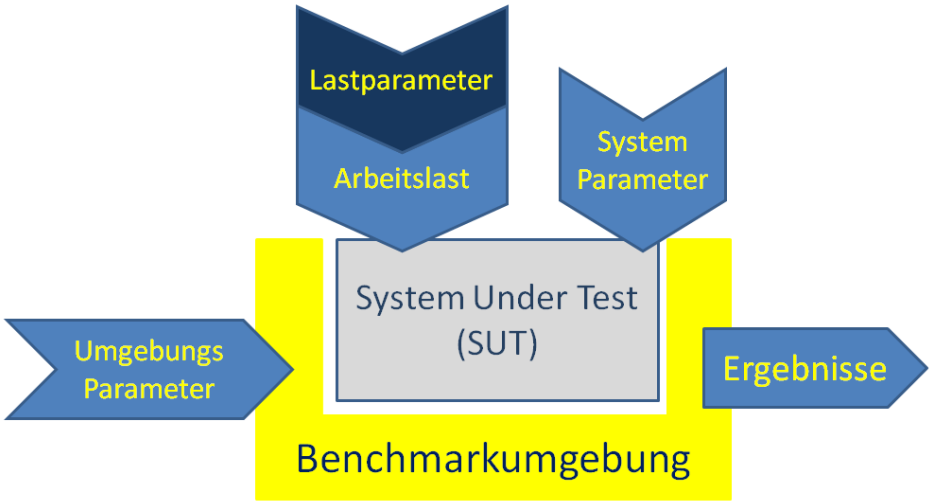



Abbildung 6.1: Übersicht über die verwendete Benchmarkmethodik

umgesetzt werden, dass sie ausschließlich das zugrundeliegende Szenario widerspiegeln.

## 6.4 Methodik

Die in dieser Arbeit verwendete Benchmarkmethodik basiert auf den Ergebnissen der Forschergruppe *QuaP2P* ([1, 62, 34, 35, 36, 33]). Danach sind folgende Komponenten für einen Benchmark (siehe Abbildung 6.1) notwendig:

**Benchmarkumgebung:** Die Benchmarkumgebung ist das Rahmenwerk, sie steuert den Ablauf des Benchmarks. Anhand der Umgebungsparameter können die Wahl der Test-Prozduren, die Art des Lastgenerators und die eingesetzten Metriken festgelegt werden.

**System under Test (SUT):** Ein SUT wird an zwei Schnittstellen in den Benchmark integriert. Zum einen muss das SUT mit der Last des Lastgenerators belegt werden können, zum anderen muss die Benchmarkumgebung in der Lage sein, Systemparameter automatisch wählen zu können. Das ist besonders bei der Optimierung von Systemen von Bedeutung.

**Arbeitslast:** Variationen der Arbeitslast werden von der Benchmarkumgebung über die Lastparameter eingestellt.

**Ergebnisse:** Die Ergebnisse werden anhand geeigneter Messfunktionen, der Metriken ermittelt. Aufgrund der Neutralität erfolgen die Messungen, wenn möglich, von außerhalb des SUT.

**Umgebungsparameter:** Je nach Anforderung werden die verwendeten Metriken und entsprecheden Testprozeduren als Umgebungsparameter festgelegt.

Eine Umsetzung dieser Komponeneten wir im folgenden Abschnitt beschrieben.

## 6.5 Implementierung

### 6.5.1 Festlegung des Szenarios

Der erste Schritt bei der Erstellung eines Benchmarks ist die Definition des Szenarios. Es definiert die Klasse der SUTs und wirkt sich auf die Wahl der Metriken und die Lastgenerierung aus. Im Fall von Netzwerkarchitekturen für MMOGs beinhaltet das zwei Entscheidungen, da Claypool et al. [12] gezeigt haben, dass die erforderliche Leistung einer Netzwerkinfrastruktur 1. von der Art eines Spiels und 2. der jeweiligen Aktion innerhalb des Spiels abhängt. Als Art des Spiels wurde in dieser Arbeit *Shooter* festgelegt, da diese Spiele eine hohe Präzision und eine kurze Deadline der Aktionen fordert. Dieses Genre kann daher als höchste Anforderung an ein P2P-Overlay gesehen werden. Die zweite Entscheidung betrifft die Aktionen innerhalb des Spiels. Da ein grundlegender Benchmark entworfen werden soll, gilt es, eine minimale Menge an Aktionen zu finden, die für das Erzeugen einer verteilten virtuellen Welt notwendig sind. Basierend auf der Unterteilung von Kapitel 2 können daher *Interest Management* und *Game Event Dissemination* als Basisfunktionalitäten festgelegt werden. Somit wird die Qualität eines Overlays durch die Genauigkeit der Nachbarlisten (Leistung des IM) und die Verzögerung der Spielereignisse (Leistung der GED) ermittelt.

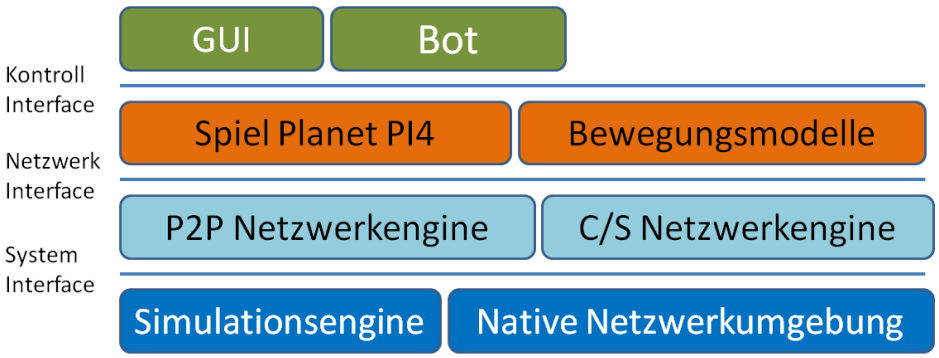



Abbildung 6.2: Übersicht über die Systemarchitektur der Benchmarkumgebung

### 6.5.2 Benchmarkumgebung

Als Basis für die Implementierung der Benchmarkumgebung wurde naheliegenderweise das Spiel *PlanetPI4* (siehe Kapitel 3) verwendet. Abbildung 6.2 zeigt eine abstrakte Sicht der Softwarearchitektur. Der modulare Aufbau von Planet PI4 ermöglicht es dabei, innerhalb einer Schicht einzelne Komponenten auszutauschen.

So kann durch das Kontrollinterface die Kontrolle der Spieler sowohl von realen Spielern als auch von der Implementierung einer künstlichen Intelligenz übernommen werden. Das Netzwerkinterface ermöglicht es, unterschiedliche Netzwerkimplementierungen zu nutzen, und das Systeminterface erlaubt sowohl einen realen Spiel- als auch einen Simulationsmodus. Im einzelnen wurden die folgende Komponenten entworfen und implementiert:

#### GUI - KI-Implementierungen

Das *Graphical User Interface* gibt die Möglichkeit, als Spieler aktiv an einer Simulation teilzunehmen. Dabei kann zum einen die Leistungsfähigkeit eines Netzwerkes in verschiedenen Mehrspielermodi getestet werden, zum anderen können Messungen von realen Spielesessions durchgeführt werden.

Als weitere Möglichkeit, dynamische Spielereignisse zu erzeugen, wurden unterschiedliche Versionen autonomer Spieler (Bots) implementiert:

**Skriptbasierte KI:** Eine skriptbasierte KI ist die einfachste Form eines Bots. Das Spielerverhalten wird direkt über eine Abfolge von Skriptbefehlen

vorgegeben.

**Zustandsorientierte KI:** Eine zustandsorientierte KI fällt Entscheidungen über mögliche Handlungen basierend auf dem aktuellen Zustand. Zustände und Übergangswahrscheinlichkeiten werden durch eine *Finite State Machine* (FSM) realisiert.

**Zielorientierte KI:** Im Gegensatz zu einer zustandsorientierten KI verwendet eine zielorientierte KI wünschenswerte Zustände als Auswahlkriterium für zukünftige Handlungen. Ziele und eine entsprechende Bewertungsfunktion können anhand eines *Behavior Trees* (BT) implementiert werden.

Alle drei KI-Varianten sind nach Champandard [10] gleich mächtig, dennoch unterscheiden sie sich deutlich in der Anwendbarkeit. So können Skripte nur für einfache Verhaltensmuster genutzt werden, da ohne die Verwendung einer logischen Struktur der Aufwand für die Umsetzung komplexerer Strategien unverhältnismäßig stark anwächst. Dagegen nutzen FSM und BT logische Strukturen, um das Spielerverhalten in wiederverwendbare Klassen einzuteilen. Dabei hat sich gezeigt, dass zielorientierte Verfahren skalierbarer sind, da die Menge der möglichen Zustände in einem Spiel beliebig groß werden kann, die verfolgten Ziele hingegen begrenzt bleiben. Genauere Untersuchungen dazu können [10] entnommen werden.

Allgemein lassen sich einer Simulation beliebig viele GUI- und Bot-Instanzen hinzufügen. Bei den Bot-instanzen kann zusätzlich auf eine grafische Ausgabe verzichtet werden, um Hardwareressourcen einzusparen.

### Spiel Planet PI4 - Bewegungsmodelle

Das Spiel Planet PI4 bietet die Möglichkeit, sämtliche netzwerkrelevanten Komponenten einer virtuellen Welt zu erzeugen. Dazu gehört eine dreidimensionale Welt, Raumschiffe, Objekte wie Asteroiden, Points of Interest wie Basen oder Upgrade-Felder und Interaktionmöglichkeiten mit anderen Spielern wie Schüsse. Auf alle Komponenten kann von der Simulation zugegriffen werden. So können beispielsweise Verfahren, die Verbindungen abbauen, wenn Spieler sich aus der gegenseitigen Sichtweite entfernen, die benötigten Informationen direkt auslesen.

Neben dem Spiel gibt es eine weitere Möglichkeit, Last für die jeweiligen Netzwerke zu erzeugen. Anhand der Bewegungsmodellkomponente können einfache Bewegungsmodelle wie zum Beispiel Random Waypoint genutzt werden, um einfache Untersuchungen an den Netzwerkimplementierungen vorzunehmen. Dabei besteht ein Spieler nur aus einem Punkt im dreidimensionalen Raum. Es gibt keine virtuelle Welt und keine Interaktionsmöglichkeit. Da keinerlei grafische Berechnungen oder aufwändige Operationen wie die Kollisionserkennung durchgeführt werden müssen, können anhand der Bewegungsmodelle höhere Spielerzahlen simuliert werden, als mit der Spieleimplementierung.

### P2P-Netzwerkengine - C/S-Netzwerkengine

Die P2P-Netzwerkengine enthält die konkrete Implementierung des P2P-Overlays, das zu bewerten ist. Um die modulare Architektur zu gewährleisten, wurde ein Klasseninterface erstellt, das von dem jeweiligen Ansatz (zum Beispiel pSense) implementiert werden muss. Das Interface selbst einthält nur die grundlegenden Netzwerkfunktionen wie beispielsweise den Verbindungsauf- und -abbau oder das Senden und Empfangen von Updatenachrichten. Weitere Strukturen wie beispielsweise die Verwendung eines Voronoidiagramms oder einer Area of Interest gehören nicht dazu. Derzeit gibt es zwei Implementierungen der P2P-Engine, die dieser modularen Architektur entsprechen (pSense und Bubblestorm).

Zusätzlich zu der P2P-Netzwerkengine kann das Spiel auch in einem klassischen Client-/Server-Modus betrieben werden.

### Simulationsengine - Native Laufzeitumgebung

Auf der untersten Ebene kann zwischen einem nativen und Simulationsmodus gewählt werden. Die Simulationsengine ist dabei in der Lage, alle Spielefunktionen als ereignisbasierte Simulation umzusetzen. D.h., Funktionen, die eine fortlaufende Ausführung erfordern, werden nicht in einer *mainloop* realisiert, sondern müssen sich bei einer speziellen Task-Engine anmelden. In der nativen Laufzeitumgebung wird dagegen eine Echtzeit-Ereignisloop verwendet. Alle zufallsbasierten Ereignisse werden dabei sowohl im Simulationsmodus als auch im Echtzeitmodus durch einen *pseudorandom number generator* (PRNG) erzeugt und lassen sich daher exakt reproduzieren.

Sämtliche Messungen werden von einem simultan laufenden Monitoringserver protokolliert, der über globales Weltwissen verfügt und die Referenzzeit vorgibt. Die dabei verwendeten Metriken werden im nächsten Abschnitt vorgestellt. Insgesamt bietet die Benchmarkumgebung eine vollständig kontrollierbare Testumgebung, die reales, simuliertes und gemischtes Spielen ermöglicht.

#### Zusätzliche Netzwerkkomponenten

Neben verschiedenen Interfaces für das SUT (Netzwerkverbindungen, IM, GED) wurden zusätzliche Netzwerkdienste implementiert, die für das Ergebnis des Benchmarks nicht relevant sind, jedoch für den Betrieb im realen Spielmodus benötigt werden.

**Objektmanagement:** Neben dem reinen Verbreiten von Updatenachrichten ist das Verändern von persistenten Objekten ein wichtiger Netzwerkdienst. Dazu werden *aktive Objekte* verwendet, die eine eigene Überwachungsfunktion nutzen, um objektspezifische Operationen durchführen zu können.

**Inhaltorientiertes publish/subscribe:** Für Teamkommunikation und objektspezifische Updates wurde ein inhaltsorientierter Publish-/Subscribe-Mechanismus implementiert.

**Globale Statistik:** Sämtlichen Spielstände oder Punkte der einzelnen Spieler werden über das globale Statistikinterface kommuniziert.

## 6.6 Metriken

Metriken sind Funktionen oder Algorithmen, die einzelne Systemmerkmale auf einen Qualitätswert abbilden. Um die gesamte Leistungsfähigkeit eines Systems erkennen zu können, gilt es, eine geeignete Menge von Metriken zu definieren, die in der Lage ist, leistungsrelevante Systemmerkmale in einem möglichst breiten Spektrum zu erfassen. Eine vollständige Leistungserkennung ist in der Regel nicht möglich, da Benchmarks aufgrund der komplexen Struktur und der dynamischen Entwicklung der verwendeten Systeme nicht analytisch vorgehen können. D.h., die Leistung eines Systems wird nicht berechnet,

sondern durch Messungen bewertet. Um dennoch ein aussagekräftiges Ergebnis zu erhalten, gilt es, die Menge der verwendeten Metriken so präzise wie möglich aus dem Szenario abzuleiten.

Im folgenden Abschnitt wird daher untersucht, wie die relevanten Dienste in einer konkreten Anwendung implementiert sind. Im einzelnen ergeben sich daraus Nachbarschafts-, Positions-, Verzögerungs- und Verbindungsmetriken.

### 6.6.1 Herleitung der Metriken

Die Definition des Szenarios hat ergeben, dass die beiden Dienste Interest Management und Game Event Dissemination die Grundlage für den Benchmark bilden. Um zu erkennen, welche Metriken für eine Bewertung eingesetzt werden können, ist es daher notwendig, die Arbeitsweise der beiden Dienste genauer zu untersuchen. Ein erster Blick legt die Vermutung nahe, dass beide Dienste anhand einer einzigen Metrik beurteilt werden können, da Positionsänderungen von Spielern auch als Ereignisse in der Spielwelt interpretiert werden können. Somit wären beide Dienste abgedeckt, wenn man als gemeinsame Metrik die Verzögerung von Ereignisnachrichten einsetzen würde. Bei dieser Vorgehensweise ergeben sich allerdings zwei Probleme: Erstens wird die aktuelle Position eines Spielers vom IM lediglich für die Erzeugung des Overlays verwendet. Die Nutzung für die Anzeige der Position im Spiel ist eigentlich nur ein Nebenprodukt. D.h., die Leistung des IM besteht hauptsächlich aus der aktuellen und korrekten Verwaltung der Nachbarschaftsliste. Zweitens werden Bewegungen im Spiel und damit vebundene Positionsänderungen in diskreter Form durch Updatenachrichten übertragen. Diese können interpoliert bzw. extrapoliert werden, um eine flüssigere Darstellung zu erreichen, die robuster gegenüber Netzschwankungen ist (Dead Reckoning). Die Verwendung eines deratigen Kompensationsverfahrens hat somit einen Einfluss auf die Qualität eines Overlays, insbesondere, wenn Overlays diesen Vorteil nutzen, um damit Ressourcen einzusparen. So verwendet beispielsweise Donnybrook eine spezielle *guided AI*, um Bewegungen von Spielern abzuschätzen. In bestimmten Fällen wird dadurch das Updateintervall auf bis zu drei Sekunden angehoben. Verzögerungen von Paketen wirken sich bei der Verwendung einer derartigen Kompensation anders aus als bei einer direkten Übermittlung der Positionen, vor allem, weil das Versenden der Updates und die Kompensationsalgorithmen

zeitlich entkoppelt sind. Daher ist eine Vergleichbarkeit durch die Messung der Verzögerung von Positionsupdates nicht gegeben. Beide Probleme lassen sich durch eine etwas spezifischere Wahl der Metriken lösen.

1.) Um die Qualität des IM messen zu können, ist eine zusätzliche Nachbarschaftsmetrik notwendig.

2.) Um die Leistung von Verzögerungskompensationsalgorithmen in den Benchmark einzubeziehen, ist es notwendig, nicht die Verzögerungen direkt zu messen, sondern deren Auswirkungen. Für Positionsupdates bedeutet das die Abweichung der angezeigten von der tatsächlichen Position (nach Verzögerungskompensation).

Die Verbreitung von Spielereignissen, die keine Verzögerungskompensation nutzt, kann direkt durch die Messung der Paketverzögerung beurteilt werden.

Neben der Qualität der lokalen Nachbarschaftsliste spielt die globale Struktur des Overlays ebenso eine wichtige Rolle für die Qualität des IM. Dafür wird eine Metrik benötigt, die in der Lage ist, eine Aussage über den Zusammenhang eines Overlays zu machen.

Abschliessend wird eine lokale Metrik für die Netzwerklast eines Knotens benötigt. Metriken über CPU-/GPU-Verbrauch, wie sie bei Grafikkartenbenchmarks und Sensornetzbenchmarks üblich sind, werden nicht verwendet, da sie für die Qualität eines Gamingoverlays von geringer Bedeutung sind[2].

### 6.6.2 Definitionen

Für die Definition der Metriken werden folgende Symbole verwendet:

- $P(t)$ ist die Menge aller Peers, die an einem Overlay zum Zeitpunkt $t$ teilnehmen.
- $T$ ist die globale Menge aller Zeitpunkte $t$.
- $vis(p)$ ist die Sichtweite (= Radius der AOI) eines Peers.
- $pos(p, q, t)$ ist die Position von Peer $p \in P(t)$, die von Peer $q \in P(t)$ zum Zeitpunkt $t$ wahrgenommen wird.

[2]Aufgrund des Szenarios kann davon ausgegangen werden, dass die Knoten hinreichend große Energie- und Prozessorleistung haben, da das Rendern einer 3D-Welt deutlich höhere Anforderungen als das Verwalten eines Overlay stellt.

- $pos(p, p, t) = pos(p, t)$ ist die tatsächliche Position von $p$ zum Zeitpunkt $t$.

- $N(p, t) = \{q \in P(t) | \Delta pos(p, q, t) \leq vis(p)\}$ ist die tatsächlich Menge der Nachbarn eines Peers $p$ zum Zeitpunkt $t$, wobei $\Delta pos(p, q, t) = \|pos(p, q, t) - pos(q, p, t)\|_2$.

- $M(p, t)$ ist die Menge der bekannten Nachbarn eines Peers $p$ zum Zeitpunkt $t$.

- $wcc(t)$ ist die größte Teilmenge eines Verbindungsgraphen, für die gilt, dass alle Knoten miteinander verbunden sind. Das entspricht der größten Partition.

### 6.6.3 Nachbarschaftsmetrik

In eine Nachbarschaftsliste werden alle Knoten eingetragen, zu denen eine direkte Verbindung besteht[3]. Sie dient als Basis für alle Updatenachrichten. Daher muss die Verwaltung der Liste schnell und zuverlässig arbeiten. Gelingt dies nicht, werden einerseits Spieler nicht erkannt und dargestellt. Das führt zu Inkonsistenzen im Spiel, die das Spielvergnügen verringern oder sogar von unfairen Spielern gezielt eingesetzt werden können, um sich beispielsweise anderen Spielern unentdeckt zu nähern. Andererseits kann es passieren, dass unnötige Nachrichten an Knoten außerhalb der AOI gesendet werden. Das erzeugt unnötig hohe Kosten bei einem Knoten.

Um ein Gütemaß für Qualität zu entwickeln, kann die Aufgabe des IM als Klassifikation betrachtet werden. Für jeden gegnerischen Knoten gilt es, zwischen den beiden Klassen „Nachbar“ und „kein Nachbar“ zu entscheiden. Bei dieser Entscheidung können zwei Fehler gemacht werden: Entweder ist ein Knoten in der Liste , obwohl er es nicht sein sollte, oder er wird nicht erkannt, obwohl er sich in der AOI befindet.

Allgemein ergeben sich somit vier Möglichkeiten zu klassifizieren:

[3]Es gibt Overlays, die aus Sicherheitsgründen keine direkte Verbindungen verwenden, sondern über Zwischenknoten kommunizieren, um die IP-Adresse des tatsächlichen Kommunikationspartners zu verbergen. Diese unterscheiden sich jedoch nicht in der Bedeutung der Nachbarschaftsliste.

| | $q \in M(p,t)$ | $j \notin M(p,t)$ |
|---|---|---|
| $q \in N(p,t)$ | true positive, $tp(p,t)$ | false negative, $fn(p,t)$ |
| $q \notin N(p,t)$ | false positive, $fp(p,t)$ | true negative, $tn(p,t)$ |

Tabelle 6.1: Konfusionsmatrix

- *Richtig positiv:* Ein Spieler befindet sich in der AOI und ist in der Nachbarschaftsliste .
- *Falsch negativ:* Ein Spieler befindet sich in der AOI und ist nicht in der Nachbarschaftsliste.
- *Falsch positiv:* Ein Spieler befindet sich außerhalb der AOI und ist in der Nachbarschaftsliste.
- *Richtig negativ:* Ein Spieler befindet sich außerhalb der AOI und ist nicht in der Nachbarschaftsliste.

Dies lässt sich auch anschaulich in der Konfusionsmatrix 6.1 darstellen.

Um nun ein Maß für die Qualität des Klassifikators zu erhalten, kann man verschiedene relativen Häufigkeiten der Kenngrößen bestimmen. Sie können auch als bedingte Wahrscheinlichkeiten interpretiert werden. Im Falle der Nachbarschaftsbestimmung sind dabei zwei Werte von Bedeutung.

- $P(\text{positiv erkannt} \mid \text{tatsächlich positiv}) = \frac{r_p}{r_p+f_n} = \text{recall}$
- $P(\text{richtig positiv erkannt} \mid \text{positiv erkannt}) = \frac{r_p}{r_p+f_p} = \text{precision}$

Einzeln betrachtet sind diese beiden Werte noch nicht besonders aussagekräftig. So könnte beispielsweise ein Overlay einen maximalen recall-Wert erzielen, wenn es jeden Knoten als Nachbarn betrachten würde. Ebenso könnte ein Overlay auf precision optimiert werden, indem kein Knoten als Nachbar klassifiziert wird. Dies sind zwar Extrembeispiele, aber sie verdeutlichen, dass beide Werte (precision und recall) nicht getrennt voneinander betrachtet werden können. Daher wird für die Bewertung eines derartigen Klassifikators gewöhnlich das kombinierte F-Maß verwendet:

$$F_\alpha = \frac{(1+\alpha) \cdot (precision \cdot recall)}{\alpha \cdot (precision + recall)} \tag{6.1}$$

Dabei wird das harmonische Mittel gebildet und $\alpha$ als Gewichtungsfaktor zwischen precision und recall verwendet ($\alpha = 2$ bedeutet, dass precision doppelt so stark gewichtet wird wie recall). Werden beide gleich gewichtet, ergibt sich:

$$F = \frac{2 \cdot (precision \cdot recall)}{(precision + recall)} \tag{6.2}$$

Dabei gilt $0 \leq F \leq 1$ mit $F = 1$ als Optimum.

Die Bewertung der Nachbarschaftsliste durch eine Interpretation als binärer Klassifikator bedeutet, dass zunächst nicht unterschieden wird, in welchem Maße falsch klassifiziert wurde. D.h. es spielt keine Rolle, ob ein Nachbar zehn Millisekunden oder fünf Sekunden zu spät erkannt wird, er wird jeweils als „falsch klassifiziert“ bewertet. Um dennoch eine aussagekräftige Bewertung zu erhalten, muss lediglich die Frequenz der Messungen hinreichend groß gewählt werden. Dadurch werden Nachbarn, die stark verzögert erkannt werden, mehrfach gezählt und damit stärker gewichtet.

Zusammenfassend lässt sich sagen, dass durch eine Interpretation als binärer Klassifikator ermittelt werden kann, wie vollständig und aktuell die Nachbarschaftslisten eines Overlays verwaltet werden. Vorraussetzung dafür ist eine hinreichend hohe Messfrequenz. Die Bewertung der Nachbarschaftsliste ist eine Makrometrik.

### 6.6.4 Positionsmetrik

Um die Genauigkeit der Spielerpositionen zu bewerten, kann der durchschnittlich Positionsfehler genutzt werden.

- Durchschnittlicher Positionsfehler - Mean Position Error (MPE):

$$err(p,t) = \frac{\sum_{q \in M(p,t)} \|pos(p,q,t) - pos(p,t)\|_2}{|M(p,t)|} \tag{6.3}$$

  Es ist zu beachten, dass diese Metrik nur die aktuell bekannten Nachbarn eines Knoten berücksichtigt. Die Positionsmetrik ist eine Mikrometrik.

### 6.6.5 Lastmetrik

$traf(p,t)$ bezeichnet die Netzwerklast eines Knotens $p$ zur Zeit $t$ in Bytes pro Sekunde. Die Netzwerklast kann als lokale Kostenmetrik betrachtet werden. Für die Leistungsbeurteilung sind Durchschnitt, Maximum und Summe/Integral von Interesse. Die Lastmetrik ist eine Mikrometrik.

### 6.6.6 Verbindungsmetrik

Die Verbindungsmetrik misst den Zusammenhang des Netzes. Dafür wird der Global Connection Component Factor (gccf) verwendet. Dieser ist wie folgt definiert:

$$gccf(t) = \frac{|wcc(t)|}{|P(t)|} \tag{6.4}$$

Der gccf ist 1.0 für den Fall, dass ein Pfad zwischen allen Knoten im Netz existiert, das Netz also nicht partitioniert ist. Im Fall einer Partitionierung gibt der gccf den Anteil der größten Partition im Netz an. Die Verbindungsmetrik ist eine Makrometrik.

## 6.7 Workload

Die letzte und gleichzeitig wichtigste Komponente beim Entwurf eines Benchmarks ist die Erzeugung der Arbeitslast. Sie bestimmt maßgeblich die Aussagekraft des Benchmarks. Folglich gilt es hierbei, Algorithmen zu finden, die sowohl reproduzierbare als auch realistische Ergebnisse liefern.

Der Designprozess lässt sich dabei in vier Abschnitte unterteilen: Zuerst muss definiert werden, für welche Klasse von Systemen die Last erzeugt werden soll. Ein entsprechendes Interface kann aus dem festgelegten Szenario abgeleitet werden. In einem zweiten Schritt können dann, basierend auf diesem Interface, Algorithmen entwickelt werden, die die Last für den Benchmark generieren. Bei dem Entwurf dieser Algorithmen ist darauf zu achten, dass sie sowohl skalierbar sind als auch reproduzierbare Ergebnisse liefern. Um auch die dritte Anforderung (realistische Last) zu gewährleisten, müssen Bewertungskriterien gefunden werden, die Auskunft über den Realitäsgrad der Last geben können. Im letzten Schritt können dann die Generierungsalgorithmen bezüglich der

Bewertungskriterien optimiert werden. Die Optimierung hängt dabei stark von den Bewertungskriterien ab und kann daher nicht automatisch erfolgen.

### 6.7.1 Definition des Interfaces

Das Interface zwischen Lastgenerator und SUT wird ebenfalls von der Wahl des Szenarios beeinflusst. D.h., es gilt, für den Lastgenerator Spielereignisse zu generieren, die Auswirkungen auf IM und GED haben. Dabei ist für den IM die zentrale Komponente die Position der Spieler. Sie definiert die lokale Nachbarschaft und veranlasst somit den Auf- und Abbau der Netzwerkverbindungen. Ein minimales Interface zwischen Lastgenerator und IM besteht daher aus den Funktionen

- Spieler $p$ tritt dem Spiel an Position $x$ bei.
- Spieler $p$ verlässt das Spiel.
- Spieler $p$ bewegt sich an Position $y$.

Dabei hängt es von der Art des Benchmarks ab, ob die Funktionen *Beitritt* und *Verlassen des Spiels* benötigt werden.

Darüber hinaus gibt es in Planet PI4 unterschiedliche Typen von Ereignissen, z.B., Schüsse, Treffer, Kills, Eroberung einer Basis oder Nutzung eines Reparaturpunktes. Diese können anwendungsspezifisch eingesetzt werden. Für einen allgemeinen Benchmark müssen diese nicht unterschieden werden. Somit ergibt sich als zusätzliche Interfacefunktion für den GED die Funktion:

- Informiere alle Nachbarn über das Ereignis $e$ (vom Typ $c$).

### 6.7.2 Formen der Lastgenerierung

Für die tatsächliche Generierung der Last können unterschiedliche Formen eingesetzt werden:

**Traces:** Traces sind Aufzeichnungen von tastächlichem Spielverhalten. Sie sind die realistischste Möglichkeit, eine Last zu generieren. Allerdings sind Traces immer auf eine feste Anzahl von Spielern beschränkt. Darüber hinaus ist es mit hohem Aufwand verbunden, Traces mit hohen Spielerzahlen zu generieren.

**Mobility Models:** MM werden häufig für die Generierung von Last eingesetzt, da sie einfach zu implementieren sind. Sie verwenden keine Spiellogik und simulieren lediglich Agenten, die sich in einer (meist leeren) Welt bewegen. Die Spielereignisse werden anhand statistischer Modelle abgeschätzt. Interaktionen können nicht in direkter Form stattfinden. D.h., sie können nur für einen Benchmark des IM eingesetzt werden. Die zwei bekanntesten MM sind *Random Walk* und *Random Waypoint*.

**KI-Spieler:** Das Ziel der Implementierung eines KI-Spielers ist es, ein realistisches Verhalten der Spieler zu simulieren. KI-Spieler sind dabei in der Lage, sämtliche Ereignistypen des Spiels zu generieren.

Aufgrund der festgelegten Teilnehmerzahl können Traces nicht für Skalierbarkeitsuntersuchungen eingesetzt werden. MM dagegen sind zwar in der Lage, das Verhalten beliebig vieler Spieler zu simulieren, können allerdings keine Interaktionen mit anderen Spielern oder der virtuellen Welt abbilden. Gerade Interaktionen sind jedoch der Ursprung der meisten Spielereignisse. Daher wurde in dieser Arbeit eine Künstliche Intelligenz zur Generierung der Arbeitslast verwendet. Aufgrund der Möglichkeit, Spielerverhalten direkt als Ziel zu formulieren, wurde eine zielorientierte KI, basierend auf *Behavior Trees* (BT), verwendet. BTs sind flexibel und skalierbar und ermöglichen im Vergleich zu Finite State Machines oder Skripten ein intuitiveres Nachbilden von Verhaltensmustern wie Angreifen, Ausweichen oder das Erobern von strategisch wichtigen Punkten.

Um dafür geeignete Parameter zu finden, ist es notwendig, die erzeugte Last zu bewerten.

### 6.7.3 Bewertung der Lastqualität

Für die Bewertung der Qualität einer erzeugten Arbeistlast kann eine reale Arbeitslast verwendet werden. D.h., es werden Traces einer realen Spielesession aufgenommen und mit der erzeugten Last des Lastgenerators verglichen. Dafür wird eine Metrik benötigt, die in der Lage ist, die Ähnlichkeit zwischen zwei unterschiedlichen Lasten zu messen [4]. Allgemein kann diese Bewertung des

[4]Veränderungen im Verhalten der Spieler, die durch eine Skalierung der Teilnehmer ausgelöst werden, können auf diese Weise nicht nachgebildet werden

Lastgenerators auch als eine Art vereinfachter Benchmark betrachtet werden. Für die Wahl des Lastgenerators und die Qualitästmetrik gilt, dass sie weder direkt noch eindeutig von dem SUT abgeleitet werden können. Vielmehr müssen sie konstruktiv entwickelt werden. Daher werden in dem folgenden Abschnitt zunächst einfache Metriken betrachtet, die dann durch komplexere Funktionen ersetzt werden. Dabei können die Upadatenachrichten, die in virtuellen Welten erzeugt werden, in zwei Kategorien eingeteilt werden: periodische Statusupdates, wie zum Beispiel Positionsupdates, und spontane Ereignisse, wie Schüsse, Treffer, etc. Da sich diese beiden Klassen deutlich unterscheiden, müssen sie getrennt voneinander betrachtet werden.

### 6.7.4 Periodische Ereignisse

Periodische Ereignisse informieren Spieler, die sich in der Umgebung befinden, über den aktuellen Status eines Spielers. Die Informationen werden dabei an alle interessierten Nachbarn gesendet.

Um für diesen Nachrichtentyp mögliche Metriken zu definieren, muss zunächst ein einfaches Modell für eine Spielesession definiert werden, das ein gegebenes Arbeitslastmodell verwendet.

$$\mathcal{G} = (P, T, \mathcal{N}, )$$

bezeichnet ein Trace einer Spielesession. Dieses Trace besteht aus einer Liste von Spielern

$$P = \{p_1, p_2, p_3, \dots\}$$

. Im Folgenden wird davon ausgegangen, dass die Spielerliste während der gesamten Spielesession konstant bleibt.

$$T = \{\tau_1, \tau_2, \tau_3, \dots\}$$

bezeichnet die Menge der Zeitpunkte, zu denen bei dem Trace Messungen erfolgt sind. Die Zeitintervalle sollten dabei eine feste Länge haben. Für den konkreten Benchmark wurde eine Länge von einer Sekunde verwendet. Die Wahl des Wertes hängt dabei von der jeweiligen Art des Spiels ab und hat einen direkten Einfluss auf das Ergebnis.

Abschließend wird die Menge der Nachbarn aller Spieler $p$ zum Zeitpunkt $\tau$ definiert durch

$$\mathcal{N} = \left(N_{p,\tau}\right),\ p = 1, \ldots, |P|,\ \tau = 1, \ldots, |T|,$$

wobei

$$N_{p,\tau} = \{q \in P \mid q \text{ ist interessiert an } p\text{'s Ereignissen zum Zeitpunkt } \tau\}$$

**Erster Ansatz: Durchschnittliche Anzahl der Nachbarn.**

Geht man davon aus, dass die Updatefrequenz der regulären Ereignisse ein Parameter des SUT ist, hat die Anzahl der Nachbarn $|N_{p,\tau}|$ den stärksten Einfluss auf die Arbeitslast. Sie bestimmt, an wieviele Knoten im Netz Updates gesendet werden.

Ein erster Ansatz besteht darin, die durchschnittliche Anzahl der Nachbarn zu verwenden:

$$\overline{N} = \frac{1}{|P||T|} \sum_{p \in P} \sum_{\tau \in T} |N_{p,\tau}|.$$

Für zwei gegebene Spielesessions $\mathcal{G} = (P, T, \mathcal{N})$ und $\mathcal{G}' = (P', T', \mathcal{N}')$ kann die Metrik definiert werden durch:

$$\mathcal{M}_{\text{mean}}(\mathcal{G}, \mathcal{G}') = |\overline{N} - \overline{N}'|.$$

Dieser Ansatz ist recht einfach und intuitiv, vernachlässigt jedoch die Varianz in der Verteilung der Nachbarn. Daher kann man zusätzlich die folgende Varianz betrachten:

$$s^2 = \frac{1}{|P||T| - 1} \sum_{p \in P} \sum_{\tau \in T} (|N_{p,\tau}| - \overline{N})^2.$$

Dies würde jedoch dazu führen, dass zwei unterschiedliche Metriken verwendet werden, deren Abhängigkeit nicht direkt ersichtlich ist. Daher wurde ein weitere Ansatz verfolgt.

**Zweiter Ansatz: Verteilung der Anzahl der Nachbarn.**

Um die Verteilung der unterschiedlichen Anzahl der Nachbarn besser ausdrücken zu können, kann man Histogramme zur Hilfe nehmen.

$$h_x(\mathcal{G}) = |\{(p,\tau) \in P \times T \mid |N_{p,\tau}| = x\}|\,,\ x \in \mathbb{N}^0$$

zählt das Vorkommen einer bestimmten Anzahl $x$ von Nachbarn. Um Sessions unterschiedlicher Länge und mit unterschiedlich vielen Teilnehmern miteinander vergleichen zu können, muss das Histogramm normalisiert werden. Das Ergebnis entspricht der empirischen Wahrscheinlichkeitsfunktion,

$$\overline{h}_x(\mathcal{G}) = \frac{h_x(\mathcal{G})}{|P| \times |T|}$$

Die entsprechende Metrik is definiert als:

$$\mathcal{M}_{\text{hist}}(\mathcal{G},\mathcal{G}') = \sum_{x \in \mathbb{N}^0} \left|\overline{h}_x(\mathcal{G}) - \overline{h}_x(\mathcal{G}')\right|.$$

**Dritter Ansatz: Übergänge.**

Ein weiterer Faktor, der bisher noch nicht in Betracht gezogen wurde, ist die Dynamik der Nachbarschaftsliste. Das Hinzufügen und Löschen von Nachbarn ist die Hauptaufgabe des Interest Managements. Das Overlay baut basierend auf den Umgebungsinformationen eines Spielers seine Topologie auf. Das bedeutet, dass die Topologie umgebaut werden muss, wenn sich die Nachbarschaft im Spiel ändert. Da diese Änderungen mit Kosten verbunden sind, spielen sie eine wichtige Rolle für die Effizienz eines Overlays. Daher ist es notwendig, dass die hohe Dynamik der Nachbarschaftsverhältnisse eines Spiels in die Generierung der Arbeitslast eines Benchmarks einfließt.

Um die Charakteristik der zeitlichen Änderungen der Nachbarschaftsliste auszudrücken, kann eine einfache Markovkette verwendet werden. Dazu geht man davon aus, dass die Anzahl von Nachbarn eines Spielers nur von der Anzahl seiner Nachbarn zu einem früheren Zeitpunkt abhängt, und ermittelt die Wahrscheinlichkeiten für alle möglichen Ereignisse (Zunahme, Abnahme und gleichbleibende Anzahl). Abhängigkeiten zweiter oder höherer Ordnung werden in diesem Ansatz ignoriert. Somit wird gezählt, wie oft sich die Anzahl

der Nachbarn von $i$ auf $j$ während der gesamten Dauer eines Spiels geändert hat .

Die entsprechende Übergangsmatrix wird definiert durch:

$$A(\mathcal{G}) = \begin{pmatrix} a_{0,0} & \cdots & a_{1,|P|} \\ \vdots & \ddots & \vdots \\ a_{|P|,1} & \cdots & a_{|P|,|P|} \end{pmatrix},$$

wobei jeder Koeffizient $a_{i,j}$ die Anzahl der Übergänge von $i$ zu je $j$ Nachbarn zählt:

$$\begin{aligned} a_{i,j} &= \left| \left\{ (p,t) \in P \times \widehat{T} \mid |N_{p,\tau_t}| = i \wedge |N_{p,\tau_{t+1}}| = j \right\} \right|; \\ \widehat{T} &= \{1, \ldots, |T| - 1\}. \end{aligned}$$

Um Sessions unterschiedlicher Länge miteinander vergleichen zu können, müssen sie normalisiert werden:

$$\overline{A}(\mathcal{G}) = \left( \overline{a}_{i,j} \right), \text{ mit } \overline{a}_{i,j} = \frac{a_{i,j}}{\sum_{k=0}^{|P|} a_{i,k}},$$

so dass $\sum_j \overline{a}_{i,j} = 1$ für alle $i \in \{0, \ldots, |P|\}$. $\overline{a}_{i,j}$ ist also die empirische Wahrscheinlichkeit eines Übergangs von $i$ zu je $j$ Nachbarn.

Somit kann der Unterschied $d_1(\overline{A}(\mathcal{G}), \overline{A}(\mathcal{G}'))$ zwischen den Nachbarschaftscharakteristiken zweier Sessions ermittelt werden, indem man die$L_1$-Norm der gemittelten Nachbarschaftsübergangsmatritzen $\overline{A}(\mathcal{G})$ und $\overline{A}(\mathcal{G}')$:

$$\begin{aligned} \mathcal{M}_{\text{trans1}}(\mathcal{G}, \mathcal{G}') &= d_1(\overline{A}(\mathcal{G}), \overline{A}(\mathcal{G}')) \\ &= \sum_{i=0}^{|P|} \sum_{j=0}^{|P|} |\overline{a}_{i,j} - \overline{a}'_{i,j}|. \end{aligned}$$

berechnet.

Bei dieser Metrik entstehen jedoch zwei Probleme. Erstens kann es Reihen in der Matrix $A$ geben, in denen alle Koeffizienten gleich null sind. In diesem Fall funktioniert die Normalisierung nicht. Eine mögliche Lösung dafür wäre, alle diese Einträge der Reihen von $\overline{A}$ auf $|P|^{-1}$ zu setzen.

Zweitens können Matrizen unterschiedlicher Größe, die beispielsweise aus Sessions mit einer unterschiedlichen Teilnehmerzahl resultieren, nicht sinnvoll miteinander verglichen werden.

**Gewichtete Übergangsmatrix.**

Um das zweite Problem lösen zu können, kann man den Histogrammansatz mit dem Markov-Übergangsmodell kombinieren.

Indem man die Reihen von $\overline{A}$ mit den empirischen Wahrscheinlichkeiten $\overline{h}$, in diesem Zustand zu sein, kombiniert, können beide Probleme gelöst werden. Vollständige Nullreihen bleiben Null, da die Wahrscheinlichkeit, in diesen Zustand zu gelangen, bei Null liegt. Matrizen unterschiedlicher Größe können nun einfach miteinander verglichen werden, indem man die kleinere Matrix mit Nullen auffüllt.

Dafür wird definiert:
$\widehat{A}(\mathcal{G}) = \left(\widehat{a}_{i,j}\right)$ mit

$$\widehat{a}_{i,j} = \overline{a}_{i,j}\overline{h}_i = \frac{a_{i,j}}{\sum_{k=0}^{|P|}\sum_{l=0}^{|P|} a_{k,l}}.$$

Abschliessend kann die Metrik definiert werden als:

$$\begin{aligned}\mathcal{M}_{\text{trans}}(\mathcal{G},\mathcal{G}') &= d_1(\widehat{A}(\mathcal{G}),\widehat{A}(\mathcal{G}')) \\ &= \sum_{i=0}^{|P|}\sum_{j=0}^{|P|} |\widehat{a}_{i,j} - \widehat{a}'_{i,j}|,\end{aligned}$$

### 6.7.5 Spontane Ereignisse

Spontane Ereignisse werden von Spielern durch gegenseitige Interaktionen wie zum Beispiel Schüsse oder Treffer erzeugt.

Das Modell der Spielesession wird dafür um irreguläre Ereignisse $\mathcal{E}$ der Klasse $\mathcal{C}$ erweitert.

$$\mathcal{G} = (P, T, \mathcal{N}, \mathcal{C}, \mathcal{E})$$

mit

$$\mathcal{E}_c = \{(t,e) \mid \text{Ereignis } e \text{ der Klasse } c \in \mathcal{C} \text{ wurde zum Zeitpunkt } t \text{ erzeugt.}\}$$

Klassen können zum Beispiel *Schießen, Treffen* aber auch *Getroffen werden* oder *Sterben* sein. Es kann angenommen werden, dass Ereignisse des gleichen

Typs die gleiche Charakteristik vorweisen und die Nachrichten die gleiche Größe haben.

In einer weiteren Repräsentation können die Ereignisse entsprechend der Intervalle $T$ der periodischen Ereignisse eingeordnet werden:

$$E_{c,\tau_i} = \{e \mid (t,e) \in \mathcal{E}_c \wedge \tau_i \leq t < \tau_{i+1}\},\ i = 1,\ldots,|T|\}$$

**Erster Ansatz: Durchschnittliche Ereignisrate**

Wiederum ist der erste Ansatz, die durchschnittliche Ereignisrate für jeden Ereignistyp zu verwenden:

$$\overline{E}_c = \frac{1}{|P||T|}\sum_{\tau \in T} |E_{c,\tau}|,$$

$$\mathcal{M}_{\text{emean},c}(\mathcal{G},\mathcal{G}') = |\overline{E}_c - \overline{E}'_c|$$

**Zweiter Ansatz: Korrelation mit der Nachbarliste.** Da die Ereignisnachrichten an alle Nachbarn geschickt werden müssen, kann es von Bedeutung sein, wie sich die Ereignisse in Abhängikeit zu der Anzahl der Nachbarn verhalten. Um das auszudrücken, kan der Korrelationskoeffizient verwendet werden:

$$r_c = \frac{\sum_{\tau \in T}(N_\tau - \overline{N})(E_{c,\tau} - \overline{E}_c)}{\sqrt{\sum_{\tau \in T}(N_\tau - \overline{N})^2 \cdot \sum_{\tau \in T}(E_{c,\tau} - \overline{E}_c)^2}},$$

Daraus folgt die Metrik:

$$\mathcal{M}_{\text{ecorr},c}(\mathcal{G},\mathcal{G}') = |r_c - r'_c|$$

### 6.7.6 Anwendung der Workloadmetriken

Basierend auf den vorgestellten Metriken wurden nun verschiedene Arbeitslasten analysiert. Als Referenzlast dienen Traces von zwei realen Spielsessions von Planet PI4 mit 8 bzw. 16 Spielern. An den Sessions nahm eine Mischung aus Anfängern und erfahrenen Shooter-Spielern teil. Die Fähigkeit der Spieler hat dabei eindeutige Auswirkungen auf die erzeugte Netzwerklast, da erfahrene Spieler einerseits mehr Interaktionen pro Sekunde erzeugten, indem sie schneller schießen und besser treffen (Treffer erzeugen ebenfalls Updatenachrichten), andereseits sind unerfahrene Spieler weniger auf Interaktionen fokussiert.

Insgesamt wurden jeweils 30 Minuten in zwei Teams gespielt. Alle Daten der Spieler wurden durch einen zentralen Monitoringserver aufgezeichnet. Die Tracefiles enthielten jeweils einen Zeitstempel, eine eindeutige Spielerkennung, die Anzahl der Nachbarn, die Anzahl der abgegebenen Schüsse, die Anzahl der Treffer und die Anzahl der Tode. Alle Werte wurden einmal pro Sekunde erhoben.

Das identische Setup wurde dann mit folgenden synthetischen Lastgenerierungen nachgestellt:

- Random Waypoint (**RWP**): Ein Bewegungsmodell, bei dem sich die Spieler an zufällige Positionen der Karte bewegen.
- Random Point of Interest (**RPOI**): Ein Bewegungsmodell, bei dem sich die Spieler zufällig an ausgewählte markante Punkte der Karte bewegen (z.B. Raumbasen, Reparaturpunkte, etc.).
- KI-Spieler mit RWP-Bewegungsmodell (**KI-RWP**): KI-Spieler, die ein RWP-Bewegungsmodell nutzen, wenn sie keine Ziele haben.
- KI-Spieler mit RPOI-Bewegungsmodell (**KI-RPOI**): KI-Spieler mit einem gesteigerten Interesse, Basen oder Reparaturpunkte anzusteuern.
- Die Aufzeichnung der realen Spielsession wird mit **REAL** bezeichnet.

Dabei wurden folgende Parameter für die Implementierung der Bots verwendet:

#### Parameter der KI-Spieler

Wie in Kapitel 6.5.2 beschrieben, wurden für Planet PI4 drei unterschiedliche Ansätze für KI-Spieler implementiert (skriptbasiert, zustandsorientert und zielorientiert). Für die Lastgenerierung wurde eine zielorientierte KI verwendet. Bei diesem Ansatz können einfache und zusammengesetzte Ziele formuliert werden, die von den Bots verfolgt werden. Zusammengesetzte Ziele bestehen aus einer Folge (hier müssen alle einfachen Ziele erfüllt sein) oder einer Auswahl (hier muss mindestens ein einfaches Ziel erfüllt sein) von einfachen Zielen, die nacheinander abgearbeitet werden. Bei einer Folge sind alle einfachen Unterziele für die Erfüllung des zusammengesetzten Ziels notwendig. Die jeweilige

Zusammensetzung der Ziele lässt sich anhand einer Baumstruktur abbilden, bei der die einfachen Ziele durch Blätter dargestellt werden. Diese stellen das Interface zur Spielwelt dar. D.h., einfache Ziele werden als Klassen implementiert, die sowohl Informationen über den aktuellen Spielzustand ermitteln als auch Aktionen direkt ausführen können. Auf diese Weise können dann einfache und komplexe Verhaltensmuster intuitiv nachgebildet werden. Die Priorität der Ziele wird periodisch, basierend auf dem aktuellen Weltzustand, neu ermittelt. Nur das zusammengesetzte Ziel (inklusive seiner Unterziele) mit der höchsten Priorität wird ausgeführt. In der Implementierung wurden vier komplexe Ziele verwendet, die auf sechs einfachen Zielen basierten. Während der Ausführung der Ziele wurden zusätzlich Kollisionsvermeidungsalgorithmen ausgeführt. Die konkreten Ziele sind folgende:

**Gehe zur Position (einfaches Ziel):** Dieses Ziel setzt die aktuelle Geschwindigkeit des Raumschiffs auf das Maximum und steuert die Zielposition an.

**Finde höchste Bedrohung (einfaches Ziel):** Das Ziel analysiert das Verhalten der Gegner, die sich in der AOI befinden. Basierend auf Distanz, Orientierung und Schussfrequenz wird dann der Gegner ausgewählt, der die größte Bedrohung darstellt.

**Greife Gegner an (einfaches Ziel):** Bei diesem Ziel wird ein Gegner verfolgt und versucht, ihn zu eleminieren. Die geeignete Strategie für die Umsetzung hängt von der Distanz zwischen Spieler und Gegner ab. Daher wurden folgende drei Strategien implementiert:

Ist der Gegner nicht in Schussweite, wird er mit maximaler Geschwindigkeit verfolgt. Ist der Gegner in Reichweite, wird er mit halber Geschwindigkeit verfolgt und zusätzlich beschossen. Ist der Gegner sehr nahe, wird versucht, ihn zu flankieren, indem seitlich um den Gegener geflogen wird. Dabei wird er ebenfalls beschossen.

**Kampf (zusammmengesetztes Ziel):** Dieses Ziel ist eine Folge von „Finde höchste Bedrohung“ und „Greife Gegner an“

**Finde Reparaturpunkt (einfaches Ziel):** Wählt den nächsten Reparaturpunkt innerhalb der AOI aus.

**Repariere Schiff (zusammmengesetztes Ziel):** Dieses Ziel ist einer Folge von „Finde Reparaturpunkt“ und „Gehe zur Position“

**Finde Basis (einfaches Ziel):** Das Ziel überprüft alle Basen innerhalb der AOI und wählt die lohnenste aus. Die Entscheidung basiert dabei auf Distanz und Status der Basen. Basen, die von Gegnern kontrolliert werden, sind lohnender als neutrale.

**Nimm Basis ein (zusammengesetztes Ziel):** Dieses Ziel ist eine Folge von “Finde Basis” und “Gehe zur Position”.

**Finde Wegpunkt (einfaches Ziel):** Wählt einen Punkt aus, der erkundet werden soll. Die Punkte werden dabei entweder durch ein random waypoint- oder ein random point of interest-Verfahren ausgewählt.

**Erkundung (zusammengesetztes Ziel):** Dieses Ziel ist eine Folge von „Finde Wegpunkt“ und „Gehe zur Position“. Es dient als Standardziel.

Die Priorisierungsfunktionen sind in Tabelle 6.2 aufgelistet.

### Vergleich der generierten Arbeitslasten

Zunächst wurden Messungen von periodischen Ereignissen untersucht. Einen ersten Eindruck über die Anzahl der Nachbarn liefert ein exemplarischer Histogrammvergleich zwischen realer und KI-RPOI-Last (siehe Abbildung 6.3).

Abbildung 6.4 zeigt zusätzlich die Übergangswahrscheinlichkeiten der realen Session $\widetilde{A}$ (mit auf 1 normalisierten Reihen) und $\widehat{A}$ (mit allen Koeffizienten auf 1 normalisiert). Um möglichst realistische Ergebnisse zu erhalten, wurde jeweils versucht, die synthetischen Lastmodelle anhand verschiedener Parameter an die reale 16-Spieler-Session anzupassen. So wurde für die Bewegungsmodelle die effektive Größe der Welt derart angepasst, dass die durchschnittliche Anzahl der Nachbarn bestmöglich mit der realen Session übereinstimmte. Dabei zeigten die Bewegungsmodelle tendenziell ein lineares Verhalten. Im Gegensatz dazu war es nicht möglich, die Parameter der KI-Spieler so anzupassen, dass gezielt die vorgegebenen durchschnittlichen Nachbarschaften erreicht wurden.

Abbildung 6.5 zeigt die Unterschiede der vier Lastgeneratoren bezüglich der Metriken $\mathcal{M}_{\text{mean}}$, $\mathcal{M}_{\text{hist}}$ und $\mathcal{M}_{\text{trans}}$. Die Bewegungsmodelle RWP und RPOI

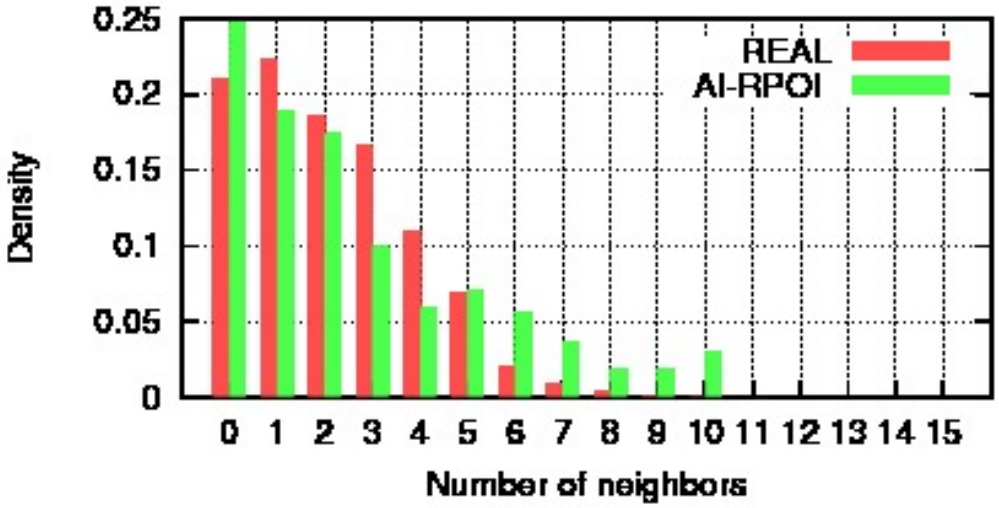
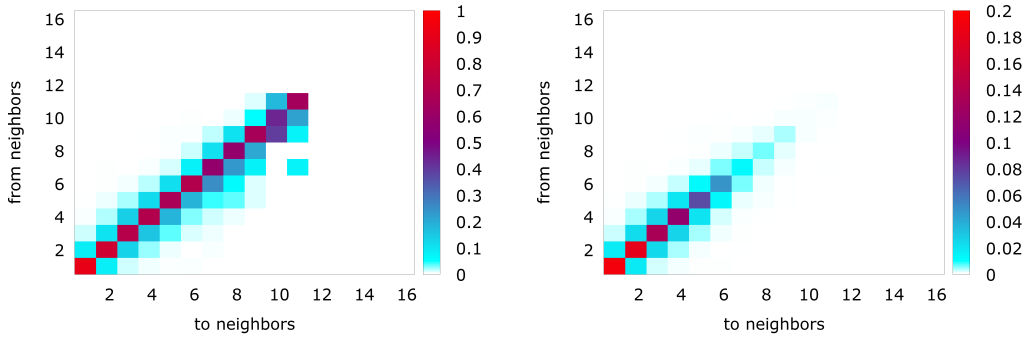



Abbildung 6.3: Visualisierung der Nachbarschaftshistogramme $\widetilde{h}$ von zwei Spielesessions mit jeweils 16 Spielern (real und KI-RPOI)




(a) normalisierte Reihen (b) normalisierte Koeffizienten

Abbildung 6.4: Visualisierung der Übergangswahrscheinlichkeiten zwischen den Nachbarn einer realen 16-Spieler Session. Insgesamt kam es zu maximal elf Nachbarn.

| Ziel | Priorisierungsfunktion |
|---|---|
| Kampf | $(\frac{CSV}{MSV}) * (\frac{VR}{DTE}) * T_{Combat}$ |
| Repariere Schiff | $(\frac{MSV-CSV}{MSV}) * (\frac{VR}{DTR}) * T_{Repair}$ |
| Nimm Basis ein | $(\frac{BVR}{CB}) * (\frac{VR}{DTB}) * T_{Capture}$ |
| Erkundung | $T_{Explore}$ |

$CSV$ : Aktueller Schildwert

$MSV$ : Maximaler Schildwert

$DTE$ : Distanz zum nächsten Gegner

$DTB$ : Distanz zur nächsten Basis

$DTR$ : Distanz zum nächsten Reparaturpunkt

$VR$ : Sichtweite

$CB$ : Eroberte Basen

$BVR$ : Basen in Sichweite

Tabelle 6.2: Priorisierungsfunktion der zusammengesetzen Ziele. Die $T$-Werte dienen als Möglichkeit, verschiedene Ziele zu priorisieren.

sind bezüglich der Metrik $\mathcal{M}_{\text{mean}}$ dem realen Spiel sehr ähnlich, bezüglich der präziseren Metriken zeigen sie dagegen deutliche Unterschiede. Ursache dafür ist, dass sie das Spielerverhalten auf eine primitive Art nachbilden. Dadurch lassen sich einfache Merkmale, wie die durchschnittliche Anzahl der Nachbarn, direkt durch die Größe der Welt manipulieren. Komplexe Verhaltensmuster der Spieler lassen sich jedoch nicht nachbilden.

Die KI-Modelle haben dagegen eine deutlich höhere Abweichung in $\mathcal{M}_{\text{mean}}$, da durch ihre komplexe Struktur kein gezieltes Anpassen möglich ist (das Anpassen der KI-Spieler geschieht durch die Priorisierungsfunktionen der Ziele, die allerdings keine direkte Auswirkung auf die Anzahl der Nachbarn haben). Bezüglich der Metriken $\mathcal{M}_{\text{hist}}$ und $\mathcal{M}_{\text{trans}}$ sind sie dagegen der realen Spiel-

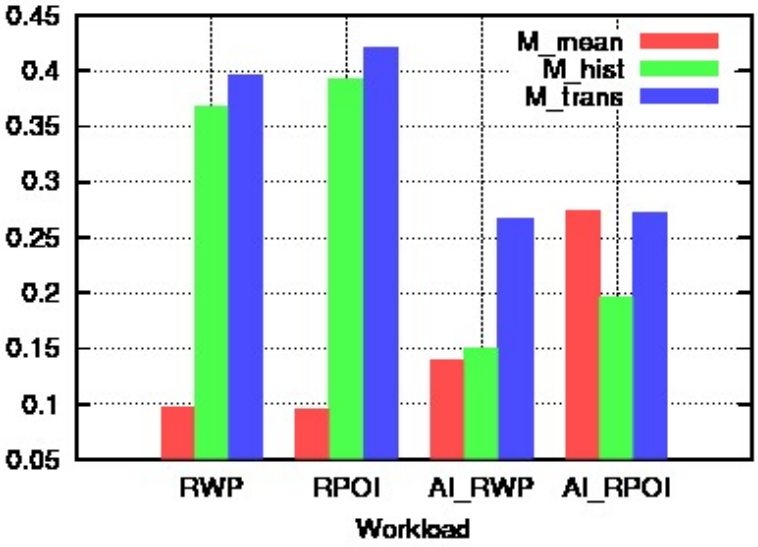



Abbildung 6.5: Unterschiede der synthetischen Lastgeneratoren. Dabei wurden die Workloadmetriken $\mathcal{M}_{\text{mean}}$, $\mathcal{M}_{\text{hist}}$, und $\mathcal{M}_{\text{trans}}$ verwendet.

situation sehr viel näher. Ursache dafür ist, dass die KI-Spieler versuchen, das tastächliche Spielerverhalten nachzubilden. Dadurch entsteht bezüglich der komplexeren Metriken ein realistischeres Verhalten.

Abschließend wurde die Skalierung der unterschiedlichen Lastgeneratoren untersucht. Abbildung 6.6 zeigt die durchschnittliche Anzahl der Nachbarn in Abhängigkeit von der Gesamtspielerzahl. Für eine weitere Auswertung werden jedoch Traces mit mehreren Spieler benötigt.

### 6.7.7 Interaktionen

Neben der Anzahl der Nachbarn ist die Nachbildung der Interaktionen die zweite wichtige Komponente realistischer Arbeitslast. Dabei generieren in Planet PI4 Schüsse, Treffer, und Kills den Großteil der Updatenachrichten. Um einen Eindruck der erzeugten Last zu bekommen, wurden alle drei Klassen sowohl von der realen Session als auch von den Bot-Versionen gemessen (siehe Tabelle 6.3). Aufgrund der fehlenden Spiellogik konnten für die Bewegungsmodelle keine Werte ermittelt werden. Der Tabelle kann entnommen werden, dass die Bot-Implementierungen aggressiver als die realen Spieler agierten. Zusätzlich haben die KI-Spieler eine höhere Treffgenauigkeit, was eine höhere Killrate zur Folge hatte. Aufgrund der zielorientierten Handlungsweise der Bots las-

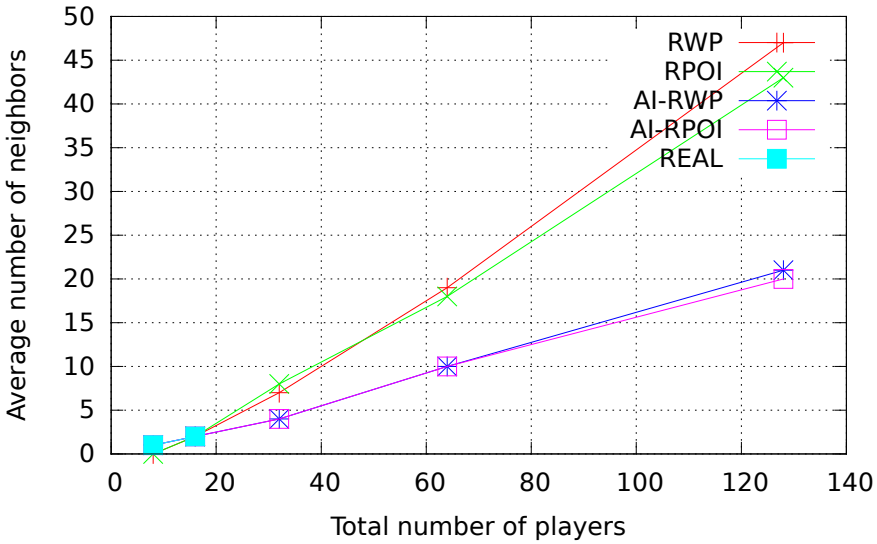



Abbildung 6.6: Durchschnittliche Anzahl der Nachbarn bei konstanter Größe der Welt und Anzahl der POI.

| Session | Schüsse | Treffer | Kills | Treffgenauigkeit |
|---|---|---|---|---|
| REAL | 148.09 | 41.76 | 0.41 | 28.2% |
| KI-RWP | 161.19 | 71.49 | 0.74 | 44.3% |
| KI-RPOI | 199.48 | 70.86 | 0.74 | 35.5% |

Tabelle 6.3: Vergleich der Schüsse/Treffer/Kills pro Minute und pro Spieler. Zusätzlich wurde die Treffgenauigkeit ermittelt.

sen sich diese Parameter nicht direkt einstellen. So hatte beispielsweise das Einstellen eines weniger aggresiven Verhaltens zur Folge, dass die Anzahl der Nachbarn sich deutlich verschlechterte. D.h., es werden zusätzliche Modelle für Aggresivität und Treffgenauigkeit benötigt, um ein realistischeres Verhalten nachzubilden.

## 6.8 Fazit

In diesem Kapitel wurde eine vollständige Benchmarkplattform für P2P-Gaming-Overlays vorgestellt. Der Benchmark wurde von Grund auf neu entworfen. Da-

für wurde die Methodik festgelegt und alle notwendigen Komponenten in einer ersten Version erstellt. Das beinhaltet sowohl die präzise Definition des Anwendungsszenarios und die Festlegung ausagekräftiger Metriken als auch die Implementierung einer skalierbaren Benchmarkumgebung und einer realistischen Lastgenerierung. Durch die modulare Softwarearchitektur der Benchmarkumgebung ist es möglich, unterschiedliche Overlaynetzwerke in die Testumgebung einzubauen, ohne dabei auf die Implementierung des Spiels, der Simulationsumgebung, der Metriken oder der Lastgenerierung zugreifen zu müssen. Anhand der hybriden Laufzeitumgebung ist es zudem möglich, Overlays live durch reales Spielen zu testen oder auch kontrollierte Simulationen durchzuführen. Als Lastgeneratoren wurden dazu zwei Bewegungsmodelle und zwei Versionen von autonomen Spielern implementiert.

Allgemein gilt für die Methodik und damit verbunden die Qualität eines Benchmarks, dass aufgrund der komplexen SUTs kein analytisches Vorgehen möglich ist. D.h., Metriken oder Arbeitslast müssen konstruktiv, basierend auf dem Wissen der Entwickler generiert werden. Vergleichbare Benchmarks, wie der Prozessorbenchmark Spec CPU [16] oder der Grafikkartenbenchmark 3DMark [14], haben dabei gezeigt, dass die Qualität durch einen evolutionären Prozess stetig verbessert werden kann. Die in diesem Kapitel beschriebene Benchmarkingplattform, kann dabei als initiale Arbeit gesehen werde, da es bisher weder verwandte Vorarbeiten noch bestehende Versionen eines derartigen Benchmarks gab.

# Kapitel 7

# Zusammenfassung

Ziel dieser Arbeit war es, zu untersuchen, wie die Qualität eines Netzwerkdienstes für MMOG-Spieleanwendungen gemessen und dargestellt werden kann und dadurch Dienste und Architekturen konstruktiv entwickelt werden können. Dazu wurde zunächst ein Überblick über die grundlegenden Techniken und Mechanismen gegeben, die für die Erzeugung einer virtuellen Welt notwendig sind. Als zentrale Dienste wurden dabei das Interest Management und die Game Event Dissemination herausgestellt. Sie sorgen dafür, dass ein zusammenhängendes Kommunikationsnetz gebildet wird, und dass alle relevanten Ereignisse der virtuellen Welt schnellstmöglich kommuniziert werden können. Um dann die Funktionsweise der einzelenen Dienste und Mechanismen genauer untersuchen zu können, wurde die prototypische Spielanwendung Planet PI4 entworfen und implementiert. Dabei lag der Fokus auf einer modularen Architektur, die ein Austauschen und Vergleichen einzelner Komponenten ermöglicht. Die Vielseitigkeit dieser Architektur wurde durch unterschiedliche Implementierungen wie der SpoVNet-Spieleanwendung oder einer Skype-basierten Version demonstriert. Für die Sprachkommunikation in verteilten virtuellen Welten wurde dann ein skalierbares Verfahren entwickelt, das auf einer hybriden Netzwerkarchitektur besteht. Es nutzt einerseits für die Signalisierung der Sprachverbindungen einen Server. Die dafür notwendigen Texnachrichten können auch für große virtuelle Welten zentral kommuniziert werden. Andererseits werden teure Audioströme dezentral zwischen den Knoten gesendet. Um Ressourcen einzusparen und zusätzlich die *immersiveness* zu verstärken wurde das Konzept der Proxemik aus der realen Welt auf die virtuelle Welt

übertragen. Insgesamt wurde somit eine skalierbarer Sprachkommunikationsdienst entworfen, der zusätzlich in der Lage ist, den Kontext der virtuellen Welt in die Sprachkommunikation miteinzubeziehen.

Neben dem Sprachkommunikationsdienst wurde auch ein verteiltes Peer-to-Peer Overlay vorgestellt, das gezielt für den Einsatz in virtuellen Welten entworfen wurde. Es basiert auf einem geografischen Publish-/Subscribe-Mechanismus, der den Aufbau der lokalen Verbindungen steuert. Für die Stabilisierung des Netzwerks wurden spezielle NetConnector-Knoten eingesetzt. Zusätzlich wurde ein geografischer Broadcastmechanismus implementiert.

Um nicht nur die Funktionsweise von unterschiedlichen Netzwerkdiensten testen zu können, sondern auch die Leistungsfähigkeit einzelner Dienste miteinander vergleichen zu können, wurde das Spiel Planet PI4 zu einer Evaluationsplattform für MMOG-Netzwerkdienste erweitert.

Dafür wurde ein vollständiger Benchmarkprozess entwickelt, der die Methodik, eine Benchmarkumgebung, sowie Metriken und unterschiedliche Möglichkeiten zur Lastgenerierung beinhaltet. Sämtliche Komponenten wurden dabei nicht nur theoretisch entworfen, sondern auch voll funktionsfähig implementiert. Sie stehen der wissenschaftlichen Gemeinschaft zur Verfügung und können für die Evaluation weitere Overlayimplementierung genutzt werden [1]. Somit wurde erstmals eine Möglichkeit geschaffen, neu Netzwerkdienste zu entwickeln und zu evaluieren, deren Qualität sich erst bei einer Teilnehmerzahl von mehreren hundert bzw. bis zu mehrern tausend Teilnehmern bermerkbar macht.

[1] `ls.wim.uni-mannheim.de/pi4/research/projects/planet-pi4/`

# Literaturverzeichnis


[1] QuaP2P Project Website. `http://www.quap2p.tu-darmstadt.de`.

[2] Shehla Abbas, Mohamed Mosbah, and Akka Zemmari. Itu-t recommendation g.114, one way transmission time. In *In International Conference on Dynamics in Logistics 2007 (LDIC 2007), Lect. Notes in Comp. Sciences*. Springer-Verlag, 1996.

[3] Dewan Tanvir Ahmed and Shervin Shirmohammadi. A dynamic area of interest management and collaboration model for p2p mmogs. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '08, pages 27–34, Washington, DC, USA, 2008. IEEE Computer Society.

[4] Helge Backhaus and Stephan Krause. Quon a quad-tree based overlay protocol for distributed virtual worlds. *Int. J. Adv. Media Commun.*, 4(2):126–139, March 2010.

[5] S.A. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. Technical Report CUCS-039-04, Computer Science Department, Columbia University, New York, NY, Sep 2004.

[6] Nathaniel E. Baughman, Marc Liberatore, and Brian Neil Levine. Cheat-proof playout for centralized and peer-to-peer gaming. *IEEE/ACM Trans. Netw.*, 15(1):1–13, February 2007.

[7] Ashwin Bharambe, John R. Douceur, Jacob R. Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan, and Xinyu Zhuang. Donnybrook: enabling large-scale, high-speed, peer-to-peer games. In *Proceedings of the ACM SIGCOMM conference on Data communication (SIGCOMM'08)*, pages 389–400, New York, NY, USA, 2008. ACM.

[8] Ashwin Bharambe, Jeffrey Pang, and Srinivasan Seshan. Colyseus: a distributed architecture for online multiplayer games. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation (NSDI'06)*, pages 12–12, Berkeley, CA, USA, 2006. USENIX Association.

[9] Paul Boustead and Farzad Safaei. Comparison of delivery architectures for immersive audio in crowded networked games. In *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '04, pages 22–27, New York, NY, USA, 2004. ACM.

[10] Alex J. Champandard. Behavior trees for next-gen game ai. In *Game Developer Conference GDC Lyon*, 2007.

[11] Jui-Fa Chen, Wei-Chuan Lin, Hua-Sheng Bai, and Shih-Yao Dai. A message interchange protocol based on a routing information protocol in a virtual world. In *AINA*, pages 377–384. IEEE Computer Society, 2005.

[12] Mark Claypool and Kajal Claypool. Latency can kill: precision and deadline in online games. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems (MMSys'10)*, pages 215–222, New York, NY, USA, 2010. ACM.

[13] The SpoVNet Consortium. Spovnet: An architecture for supporting future internet applications. In *Joint EuroNGI and ITG Workshop on Visions of Future Generation Networks (EuroView2007), July 23-24 2007*, 2007.

[14] Futuremark Corporation. 3DMark Vantage. `http://www.futuremark.com/benchmarks/3dmarkvantage/`.

[15] Standard Performance Evaluation Corporation. SPECjms2007. `http://www.spec.org/jms2007/`.

[16] Standard Performance Evaluation Corporation. SPEC's Benchmarks and Published Results. `http://www.spec.org/benchmarks.html`.

[17] George F. Coulouris and Jean Dollimore. *Distributed systems - concepts and design.* Addison-Wesley, 1988.

[18] Lu Fan, Phil Trinder, and Hamish Taylor. Design issues for peer-to-peer massively multiplayer online games. *International Journal of Advanced Media and Communication*, 4(2):108–125, March 2010.

[19] Chris GauthierDickey, Virginia Mary Lo, and Daniel Zappala. Using n-trees for scalable event ordering in peer-to-peer games. In Wu chi Feng and Ketan Mayer-Patel, editors, *NOSSDAV*, pages 87–92. ACM, 2005.

[20] D. Haage, R. Holz, H. Niedermayer, and P. Laskov. A Cross-Layer Information Service for Overlay Network Optimization. In *In 4th GI/ITG KuVS Workshop on The Future Internet and 2nd Workshop on Economic Traffic Management (ETM), Zürich*, 2009.

[21] Edward Twitchell Hall. *The Hidden Dimension.* Anchor Books, 1966.

[22] Tristan Henderson. Latency and user behaviour on a multiplayer game server. In *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*, pages 1–13, London, UK, 2001. Springer-Verlag.

[23] Shun-Yun Hu and Guan-Ming Liao. Scalable peer-to-peer networked virtual environment. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 129–133, New York, NY, USA, 2004. ACM.

[24] Shun-Yun Hu and Guan-Ming Liao. VON: A Scalable Peer-to-Peer Network for Virtual Environments. In *IEEE Network, vol. 20, no. 4, Jul./Aug. 2006*, pages 22–31, 2006.

[25] C. Hübsch, C. Mayer, S. Mies, R. Bless, O. Waldhorst, and M. Zitterbart. Reconnecting the Internet with ariba: Self-Organizing Provisioning of End-to-End Connectivity in Heterogeneous Networks. In *Proceedings of ACM SIGCOMM*, pages 131–132. ACM, August 2009.

[26] J. Huizinga. *Homo Ludens:.* Homo Ludens: A Study of Play-element in Culture. Routledge and K. Paul, 1949.

[27] Takuji Iimura, Hiroaki Hazeyama, and Youki Kadobayashi. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player

online games. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, NetGames '04, pages 116–120, New York, NY, USA, 2004. ACM.

[28] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc, 1991.

[29] Yoshihiro Kawahara, Tomonori Aoyama, and Hiroyuki Morikawa. A peer-to-peer message exchange scheme for large-scale networked virtual environments. *Telecommunication Systems*, 25(3-4):353–370, 2004.

[30] Joaquín Keller and Gwendal Simon. Solipsis: A massively multi-participant virtual world. In Hamid R. Arabnia and Youngsong Mun, editors, *PDPTA*, pages 262–268. CSREA Press, 2003.

[31] Bjorn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-peer support for massively multiplayer games. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, 2004.

[32] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28:690–691, 1979.

[33] Max Lehn, Christian Groß, and Tonio Triebel. *Benchmarking Peer-to-Peer Systems*, volume 7847 of *Lecture Notes in Computer Science*, chapter Application Benchmarks for Peer-to-Peer Systems: Peer-to-Peer Overlays for Online Games, pages 131–154. Springer, 2013.

[34] Max Lehn, Christof Leng, Robert Rehner, Tonio Triebel, and Alejandro Buchmann. An online gaming testbed for peer-to-peer architectures. *SIGCOMM Comput. Commun. Rev.*, 41(4):474–475, August 2011.

[35] Max Lehn, Tonio Triebel, Christian Gross, Dominik Stingl, Karsten Saller, Wolfgang Effelsberg, Alexandra Kovacevic, and Ralf Steinmetz. *From Active Data Management to Event-Based Systems and More*, volume 6462 of *Lecture Notes in Computer Science*, chapter Designing Benchmarks for P2P Systems, pages 209–229. Springer, nov 2010.

[36] Max Lehn, Tonio Triebel, Robert Rehner, Benjamin Guthier, Stephan Kopf, Alejandro Buchmann, and Wolfgang Effelsberg. On synthetic workloads for multiplayer online games: a methodology for generating representative shooter game workloads. *Multimedia Systems (MMSJ)*, pages 1–12, February 2014. The final publication is available at http://link.springer.com.

[37] Yi J. Liang, Eckehard G. Steinbach, and Bernd Girod. Real-time voice communication over the internet using packet path diversity. In *Proceedings of the ninth ACM international conference on Multimedia*, MULTIMEDIA '01, pages 431–440, New York, NY, USA, 2001. ACM.

[38] Martin Mauve, Jürgen Vogel, Volker Hilt, and Wolfgang Effelsberg. Local-lag and timewarp: providing consistency for replicated continuous applications. *IEEE Transactions on Multimedia*, 6(1):47–57, 2004.

[39] C. Mayer, S. Mies, O. Waldhorst, R. Holz, D. Haage, C. Blankenhorn, R. Holz, G. Koch, B. Koldehofe, and F. Lampi. Spontaneous Virtual Networks: On the Road towards the Internet's Next Generation. *it - Information Technology Special Issue on Next Generation Internet*, 50(6):367–375, December 2008.

[40] Philip Mildner. Entwurf, Implementierung und Evaluation einer echtzeitfähigen und skalierbaren Peer-to-Peer-Spieleanwendung unter Verwendung der Aribasoftware als Netzwerkinterface. Master's thesis, Universität Mannheim, 2010.

[41] Philip Mildner, Tonio Triebel, Stephan Kopf, and Wolfgang Effelsberg. A scalable peer-to-peer-overlay for real-time massively multiplayer online games. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 304–311, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[42] C. Nguyen, F. Safaei, and P. Boustead. A distributed server architecture for providing immersive audio communication to massively multiplayer online games. In *Networks 2004. (ICON 2004). Proceedings. 12th IEEE International Conference on*, volume 1, pages 170–176, 2004.

[43] Thomas Plotkowiak. Implementierung einer integrierten, distanzbasierten Sprachkommunikation f¨ur Peer-to-Peer-Spiele. Master's thesis, Universität Mannheim, 2008.

[44] Dirk Haage Ralph Holz. Clio/unisono - practical and distributed overlay-wide network measurement. *KuVS Fachgespräch Future Internet*, 2009.

[45] Laura Ricci and Andrea Salvadori. Nomad: Virtual environments on p2p voronoi overlays. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *OTM Workshops (2)*, volume 4806 of *Lecture Notes in Computer Science*, pages 911–920. Springer, 2007.

[46] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. RFC 3261: SIP: Session Initiation Protocol. Technical report, IETF, 2002.

[47] Farzad Safaei. Dice: Internet delivery of immersive voice communication for crowded virtual spaces. In *Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, VR '05, pages 35–41, Washington, DC, USA, 2005. IEEE Computer Society.

[48] F. Schiller. *Über die ästhetische Erziehung des Menschen in einer Reihe von Briefen.* 15. Brief. 1793.

[49] Arne Schmieg, Michael Stieler, Sebastian Jeckel, Patric Kabus, Bettina Kemme, and Alejandro Buchmann. pSense - Maintaining a Dynamic Localized Peer-to-Peer Structure for Position Based Multicast in Games. In *IEEE International Conference on Peer-to-Peer Computing (P2P'08)*, 2008.

[50] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rfc 3550, rtp: A transport protocol for real-time applications, 2003.

[51] Kundan Singh and Henning Schulzrinne. Peer-to-peer internet telephony using sip. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '05, pages 63–68, New York, NY, USA, 2005. ACM.

[52] Skype. A peer-to-peer internet telephony software. Webpage: `www.skype.com`.

[53] I. Stoica, R Morris, D Karger, F Kaashoek, and H Balakrishnan. Chord:Scalable Peer-to-Peer Lookup Service for Internet Applications. In *In Proceedings ACM Sigcomm 2001, San Diego, CA, Aug*, 2001.

[54] P. Svoboda, W. Karner, and M. Rupp. Traffic Analysis and Modeling for World of Warcraft. *IEEE International Conference on Communications (ICC'07)*, pages 1612–1617, June 2007.

[55] Andrew S. Tanenbaum and Maarten van Steen. *Verteilte Systeme*. PEARSON STUDIUM, 2., aufl. edition, 2007.

[56] Egemen Tanin, Aaron Harwood, Hanan Samet, Sarana Nutanong, and Minh Tri Truong. A serverless 3d world. In Dieter Pfoser, Isabel F. Cruz, and Marc Ronthaler, editors, *GIS*, pages 157–165. ACM, 2004.

[57] Wesley W. Terpstra, Jussi Kangasharju, Christof Leng, and Alejandro P. Buchmann. Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM'07)*, pages 49–60, New York, NY, USA, 2007. ACM.

[58] Wesley W. Terpstra, Christof Leng, and Alejandro P. Buchmann. Brief announcement: Practical summation via gossip. In *Twenty-Sixth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2007)*, pages 390–391, New York, NY, USA, 2007. ACM Press.

[59] Tonio Triebel, Benjamin Guthier, and Wolfgang Effelsberg. Skype4games. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 13–18, New York, NY, USA, 2007. ACM.

[60] Tonio Triebel, Benjamin Guthier, Thomas Plotkowiak, and Wolfgang Effelsberg. Peer-to-peer voice communication for massively multiplayer online games. In *Proceedings of the 6th IEEE Conference on Consumer Com-*

*munications and Networking Conference*, CCNC'09, pages 1282–1286, Piscataway, NJ, USA, 2009. IEEE Press.

[61] Tonio Triebel, Benjamin Guthier, Richard Süselbeck, Gregor Schiele, and Wolfgang Effelsberg. Peer-to-peer infrastructures for games. In *NOSSDAV '08: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 123–124, New York, NY, USA, 2008. ACM.

[62] Tonio Triebel, Max Lehn, Robert Rehner, Benjamin Guthier, Stephan Kopf, and Wolfgang Effelsberg. Generation of synthetic workloads for multiplayer online gaming benchmarks. In *NetGames '12: Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, pages 1–6, Piscataway, NJ, USA, 2012. IEEE Press.

[63] Oliver P. Waldhorst, Christian Blankenhorn, Dirk Haage, Ralph Holz, Gerald G. Koch, Boris Koldehofe, Fleming Lampi, Christoph P. Mayer, and Sebastian Mies. Spontaneous virtual networks: On the road towards the internet's next generation. *it - Information Technology*, 50(6):367–375, 2008.