ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

STL SYSTEMTECHNIKLABOR

Netcode – Networking in Online Games

Maverick Studer

Technical Report – STL-TR-2021-02 – ISSN 2364-7167

stl.htwsaar.de

Technische Berichte des Systemtechniklabors (STL) der htw saar
Technical Reports of the System Technology Lab (STL) at htw saar
ISSN 2364-7167

Maverick Studer: Netcode – Networking in Online Games
Technical report id: STL-TR-2021-02

First published: April 2021
Last revision: April 2021
Internal review: André Miede

For the most recent version of this report see: https://stl.htwsaar.de/

Title image source: TheDigitalArtist, https://pixabay.com/de/photos/erde-internet-globalisierung-2254769/

htw saar – Hochschule für Technik und Wirtschaft des Saarlandes (University of Applied Sciences)
Fakultät für Ingenieurwissenschaften (School of Engineering)
STL – Systemtechniklabor (System Technology Lab)
Prof. Dr.-Ing. André Miede (andre.miede@htwsaar.de)
Goebenstraße 40
66117 Saarbrücken, Germany
https://stl.htwsaar.de

Netcode – Networking in Online Games

Maverick Studer
htw saar – Hochschule für Technik und Wirtschaft des Saarlandes
Seminar "Angewandte Informatik"
Wintersemester 2020/2021

***Zusammenfassung*—Online Gaming ist der neuste Trend in der Gaming-Industrie. Wo Spieler über das Internet miteinander interagieren, finden komplexe Abläufe auf Netzwerkebene statt. Oft hört man in diesem Kontext das Buzzword Netcode, jedoch ist die Bedeutung des Begriffs den meisten unklar. Der Bericht stellt die verschiedenen Aspekte dar, die Netcode umfasst. Dazu werden grundlegende Kenntnisse zum Verständnis der Internetkommunikation vermittelt. Anschließend wird auf die wichtigsten modernen Netzwerkarchitekturen und Technologien zur Synchronisation zwischen den Spielern eingegangen. Dadurch wird ein umfassendes Bild vom Thema Netcode geschaffen, sowie Kenntnisse über den aktuellen Stand der Technik gewonnen.**

I. Einführung

Die Gaming-Industrie ist aktuell schon größer als die Film- und Musikbranche und wächst weiterhin exponentiell. Insbesondere die Bereiche *Mobile* und *eSports* finden bei der Generation Millennials großen Anklang [1]. Videospiele haben sich als fester Bestandteil der modernen Internetkultur entwickelt. Die Spiele werden immer sozialer und unterstützen das Spielen mit Freunden durch Multiplayer- oder Online-Funktionalität. In solchen Online Games können je nach Spiel kleine Gruppen bis hin zu tausenden Spielern interagieren. Heutzutage stellt dies eine Selbstverständlichkeit dar, jedoch ist die Technik, die ein flüssiges Zusammenspielen möglich macht, sehr komplex. Unter Netcode versteht man alles was mit *Networking* in Online Games zu tun hat. Networking befasst sich mit Technologien, um Computer miteinander zu verbinden und eine Kommunikation zu ermöglichen [2, Chapter 1 – Introduction And Overview]. Zum Aufbau eines solchen Netzwerks ist Hardware und Software notwendig. Um ein Verständnis der Besonderheiten in Bezug auf Online Gaming zu schaffen, müssen grundlegende Prinzipien der Internetkommunikation klar sein. Auf diese wird daher zu Beginn eingegangen. Anschließend werden die wesentlichen Netzwerktopologien, sowie deren Vor- und Nachteile betrachtet. In Online Games müssen ständig Daten zwischen verschiedenen Computern über das Internet ausgetauscht werden. Hierbei können diverse Probleme auftreten, wie Paketverlust, hohe Latenz und Jitter. Es wird auf die wichtigsten Störungen eingegangen, sowie deren Einfluss auf das Spielerlebnis. Abschließend werden Technologien vorgestellt, um diesen entgegenzutreten.

II. Grundlegende Informationen

Zum Verständnis der folgenden Inhalte muss kurz auf das Funktionsprinzip des Internets eingegangen werden. Dazu muss man insbesondere die Funktionsweise der Protokolle betrachten, da diese die Basis des Internets darstellen. Ebenso wird das Routing, also die Wegfindung von Paketen im Internet erläutert. Die Ressourcen, die jeder Host in einem Netzwerk zur Teilnahme benötigt, sind Bandbreite, Latenz und Rechenleistung. Diese sind insbesondere für Online Games als ressourcenintensive Internetanwendungen wichtig.

A. Basic Internet Architecture

1) Protokolle: Im Online Gaming müssen viele Daten in nahezu Echtzeit verteilt werden. Die Spiele sind Anwendungen im Internet, welche auf Dienste zugreifen. Dienste im World Wide Web basieren auf Protokollen, um einen Datenaustausch zu realisieren [3, Chapter 4 – Basic Internet Architecture]. Protokolle definieren Standards für die Übertragung und schaffen somit eine einheitliche Kommunikationsbasis [4, Chapter 2: Understanding Network Protocols and Standards].

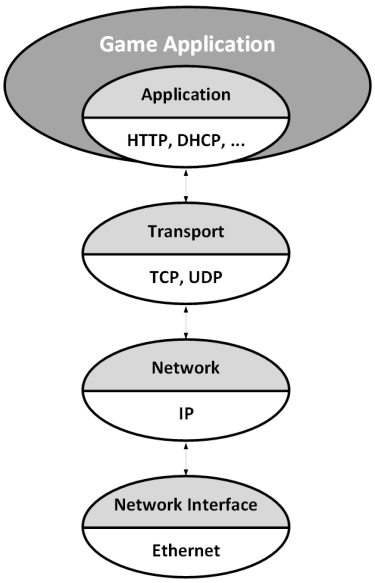

Abbildung 1. Zusammenhang Endanwendung - TCP/IP Protokollstack

Das Internet baut auf der TCP/IP-Protokollfamilie auf. Hierbei handelt es sich um ein vierschichtiges Modell, bestehend aus einer Anwendungsschicht, einer Transportschicht, einer Internetschicht und einer Netzzugangsschicht (vgl. Abbildung 1). Die Netzzugangsschicht befasst sich mit der Verknüpfung von Netzen beispielsweise durch Funk oder Kabel, bzw. Ethernet [4, Chapter 2: Understanding Network Protocols and Standards]. Auf der Internetschicht ist das *Internet*

Protocol, kurz IP, anzutreffen. IP ist für die Übertragung von Datenpaketen von Host zu Host zuständig. Dabei kann es auch eine Übertragung über Netzwerkgrenzen hinweg realisieren. Auf der Transportschicht sind die Protokolle TCP und UDP zu finden. TCP steht für *Transmission Control Protocol* und ist ein verbindungsorientiertes Protokoll. Das heißt bevor eine Übertragung erfolgen kann, muss zuerst eine Verbindung zwischen zwei Hosts aufgebaut werden. Dazu führen die Kommunikationspartner einen Drei-Wege-Handshake aus, um sich gegenseitig zu authentifizieren. Die Verbindung wird erst bei erfolgreicher und vollständiger Übertragung beendet [2, Chapter 13 – Reliable Stream Transport Service (TCP)]. Das *User Datagram Protocol* (UDP) hingegen ist verbindungslos, erspart sich somit den Mehraufwand eines Verbindungsaufbaus, kann aber auch keine Übertragung garantieren [2, Chapter 12 – User Datagram Protocol (UDP)]. In der Anwendungsschicht wird die Kommunikation der Programme realisiert.

2) Routing: Das Routing im Internet wird auch als *Next-Hop-Routing* bezeichnet. Die Router kennen nicht den gesamten Weg zum Ziel, sondern nur den nächsten Hop, also den nächsten Router. Die Netzwege sind meist so lang, dass viele Router involviert werden müssen. Die Route ist zudem nicht fest, sondern kann sich dynamisch ändern, falls Geräte ausfallen. Das Internet Protocol versucht jedoch stets den kürzesten Pfad zu finden [2, Chapter 7 – Internet Protocol: Connectionless Datagram Delivery]. Hierzu werden Algorithmen der Graphentheorie eingesetzt. Jedem Pfad werden seine Kosten zugeordnet. Diese Zuordnung erfolgt basierend auf verschiedenen Metriken, wie beispielsweise Übertragungsgeschwindigkeit, Latenz oder Datendurchsatz. Bei der Auswahl des nächsten Routers werden die Kosten aller möglichen Optionen ausgewertet [2, Chapter 14 – Routing: Cores, Peers, And Algorithms].

Je nach Übertragungsprotokoll ist es nicht ungewöhnlich, dass für eine Datenübertragung mehrere Pakete versandt werden müssen. Bei TCP muss zum Verbindungsaufbau bereits Kommunikation stattfinden. Zudem müssen erhaltene Nachrichten durch eine Antwort bestätigt werden [5]. Für das Online Gaming bedeutet dies, dass sich Verzögerungen in der Übertragung zwischen Clients selbst im Optimalfall nicht vermeiden lassen.

B. Ressourcen

Die wesentlichen Ressourcen, mit denen ein Host in einem Netzwerk ausgestattet sein muss, sind Bandbreite, niedrige Latenz und Rechenleistung.

Unter Bandbreite versteht man die Übertragungskapazität einer Verbindung. Man misst die Anzahl an übertragenen Datenpaketen in einer Zeiteinheit. Die Bandbreite bestimmt die Geschwindigkeit der Kommunikation mit anderen Hosts im Internet [6]. Die Übertragungsgeschwindigkeit ist abhängig von verschiedenen Faktoren. Datentransfertechnologien spielen eine wichtige Rolle. Die Distanz der Hosts und die Auslastung des Übertragungsmediums haben ebenfalls Einfluss auf die Übertragung. Zudem haben Datenpakete je nach Art des Übertragungsmediums eine unterschiedliche Geschwindigkeit [7]. Vergleicht man eine kabellose mit einer kabelgebundenen Verbindung ist das Ergebnis eindeutig. Eine Übertragung über Kabel ist schneller, störungsfreier und die Latenz ist geringer [8]. Aber auch bei Kabelverbindungen kann man zwischen den verschiedenen Kabelarten unterscheiden. Lichtwellenleiter, auch Glasfaserkabel ermöglichen schnellere Übertragungsraten als Kupferleitungen [4, Chapter 1: Understanding Networks].

Die Latenz, auch Ping genannt, bezeichnet die Zeit, die eine Nachricht von Sender zum Empfänger benötigt. Der einfachste Weg, um Latenz zu messen, ist der ping-Befehl [7]. Hierbei wird eine *ICMP Echo Request* zum Server versandt. Dieser antwortet mit einer *ICMP Echo Reply* [2, Chapter 9 – Internet Protocol: Error And Control Messages (ICMP)]. Es wird die Zeit zwischen Absenden der Anfrage und Erhalten der Antwort gemessen. Latenz lässt sich auch nie vollkommen eliminieren, denn selbst bei Glasfaserkabeln können sich die Daten mit maximal Lichtgeschwindigkeit bewegen. Für die meisten modernen Spiele ist bereits eine Verzögerung von mehr als 100ms als kritisch anzusehen. Jedoch ist es nicht selten, dass Spieler solche oder noch höhere Pings haben. Eine Umfrage bei Spielern mit über 18.000 Teilnehmern hat ergeben, dass über 8.000 einen Ping zwischen 80 und 100ms haben, während über 4.000 mit einem Ping über 100ms spielen [9]. Das vollständige Ergebnis der Umfrage ist in Abbildung 2 dargestellt.

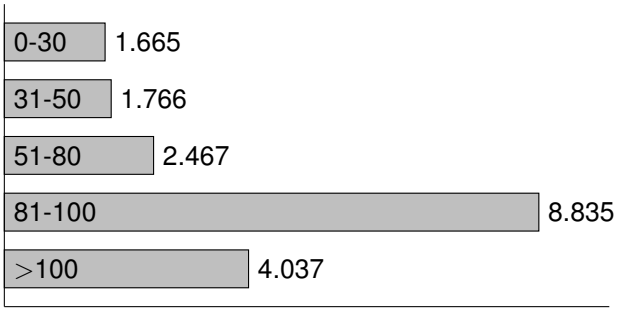

Abbildung 2. Spieler Pings - League of Legends Nordamerika (nach [9])

Die Hardware ist ebenfalls eine wichtige Ressource. Unabhängig vom Netzwerkmodell muss ein Host oder Server mit vielen anderen Hosts Daten austauschen. Die Datenpakete müssen von der CPU verarbeitet werden. Der Aufwand dieser Verarbeitung wird durch die Anzahl der Teilnehmer, Frequenz der Kommunikation und Komplexität der verwendeten Protokolle bestimmt. Je mehr Rechenleistung ein Host bereitstellt, desto schneller kann eine Datenverarbeitung erfolgen [7].

III. Netzwerkmodelle

Das Networking wird grundlegend durch die Architektur des Netzwerkes und die Rollen der Teilnehmer beeinflusst. Die für das Online Gaming am meisten verwendeten Netzwerkmodelle sind in Abbildung 3 dargestellt. Im Folgenden wird auf jedes Modell kurz eingegangen, sowie dessen Vor- und Nachteile dargestellt.

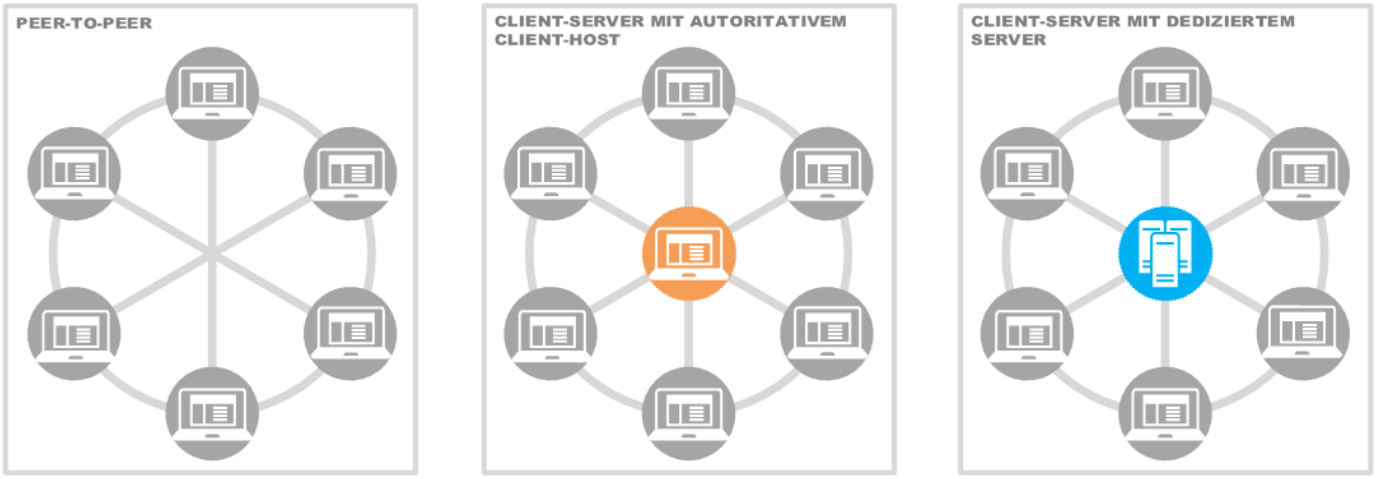

Abbildung 3. Verbreitete Netzwerkmodelle im Bereich Online Gaming (nach [7])

A. P2P

Jeder Client verwaltet sein eigenes Spielgeschehen. Alle Clients sind gleichberechtigt und kommunizieren direkt miteinander [3, Chapter 3 – Recent Online and Multiplayer Games]. Man spricht von einer vollvermaschten Netzwerktopologie, da alle Hosts miteinander verbunden sind [4, Chapter 4: Understanding Network Operating Systems]. Führt ein Client eine Operation durch, die Auswirkung auf das Spielgeschehen aller Teilnehmer hat, so muss er sein Update an alle anderen Clients senden. Diese entscheiden dann selbst, wie sie den neuen Spielstatus ermitteln und die Aktualisierung einbringen. Dieses Modell war in den Anfängen des Online Gaming weitverbreitet. Es wird heutzutage jedoch vorwiegend bei *Real-Time-Strategy-Games*, also Echtzeit-Strategiespielen, verwendet [10]. Vorteil des P2P-Modells ist die Tatsache, dass man keinen Server benötigt. Dadurch können Kosten eingespart werden. Ein Server ist teuer in der Anschaffung und im Betrieb. Ein weiterer Vorteil ist die direkte Verbindung der Clients. Dadurch, dass die Kommunikation ohne Zwischenstation ablaufen kann, sind die Netzwege kürzer. Die Kommunikation kann schneller ablaufen, falls alle Teilnehmer eine gleichermaßen gute Verbindung haben. Die Nachteile überwiegen jedoch den Vorteilen. Bei P2P gibt es keinen autoritativen Host oder Server. Jeder Client muss für sich Entscheidungen treffen. Wenn sich durchgeführte Aktionen widersprechen, ist es nicht ungewöhnlich, dass die Clients unterschiedliche Entscheidungen bezüglich der Einbringung von empfangenen Updates treffen. Dies führt oftmals dazu, dass die Clients in einen asynchronen Zustand gelangen [5]. Zudem ist P2P anfällig gegen *Cheating*. Jeder Client verwaltet sein Spielgeschehen. Es ist also möglich mit Fremdprogrammen einzugreifen und eine Manipulation durchzuführen [3, Chapter 7 – Playability versus Network Conditions and Cheats]. Da alle Clients gleichberechtigt sind, bleibt eine solche Veränderung nicht nur unerkannt, sie wirkt sich auch auf alle anderen Teilnehmer aus. Ein weiterer Nachteil stellt die Anzahl an Verbindungen dar, die aufgebaut werden müssen. Jeder Client muss mit jedem anderen Client im P2P-Netz eine Verbindung aufbauen. Dadurch steigt der Netzwerkverkehr und somit die benötigte Bandbreite enorm [7]. Zudem muss sich jeder Client selbst um Firewall-Regeln, wie beispielsweise Port-Freigaben kümmern.

B. Server-Client mit autoritativem Game-Server (dedizierter Server)

Alle Clients sind mit einem Server verbunden. Man spricht von einem dedizierten Server. Hierbei handelt es sich um einen Computer, dessen einzige Aufgabe es ist, Ressourcen und Logik für Clients über das Netzwerk bereitzustellen [4, Chapter 1: Understanding Networks]. Dieser allein verwaltet das Spielgeschehen. Die Clients senden bei Operationen Updates an den Server. Dieser entscheidet über deren Validität und sendet den resultierenden Spielstatus an alle Teilnehmer. Die Clients können hierbei selbst Berechnungen ausführen oder als reine Terminals agieren, die nur das Spielgeschehen des Servers wiedergeben [10]. Die meisten modernen Online Games nutzen dieses Architekturmodell. Seinen Anfang nahm es mit Spielen wie *Quake3* und *Half-Life* [11]. Durch die zentrale Verwaltung ergeben sich zahlreiche Vorteile. Ein Server stellt mehr Rechenleistung bereit als ein normaler Computer. Dadurch können Berechnungen schneller durchgeführt werden. Mechanismen zur Prävention von Lag, Asynchronität und Cheating können auf Serverseite implementiert werden [5]. Als Konsequenz ergibt sich ein flüssigeres Spielerlebnis und eine höhere Sicherheit der Anwendung. Zudem lassen sich dedizierte Server auch besser skalieren. Steigen Spielerzahlen, so können mehr Ressourcen allokiert werden, um dem größeren Bedarf entgegenzutreten. Die Nachteile sind ebenso zu berücksichtigen. Es ist eine aufwendige Implementierung der Serveranwendung und eine leistungsstarke Hardware notwendig. Das Hosten der Server ist in der Regel sehr kostspielig. Zudem muss bei Online Games, wo Spieler aus aller Welt zusammenkommen, eine große Flächenabdeckung erreicht werden [7]. Es ist notwendig Server an vielen Standorten rund um den Globus zu verteilen.

C. Server-Client mit autoritativem Client-Host

Bei diesem Modell schlüpft einer der Clients in die Rolle des Servers. Dieser Client nimmt ebenfalls am Spiel teil. Er verwaltet jedoch auch den Spielstatus [5]. Diese Architektur kommt mit denselben Vorteilen wie die Server-Client-Architektur, welche einen autoritativen Game-Server einsetzt. Man spart sich jedoch die Kosten für das Betreiben eines Servers. Die Host-Maschine muss jedoch über leistungsstarke Hardware und eine stabile Internetverbindung mit ausreichend Bandbreite verfügen. Zudem hat der Client-Host einen klaren Vorteil, da er keinerlei Verzögerung hat. Voraussetzung für akzeptable Latenzzeiten ist die geografische Nähe aller Clients. Ein weiterer Nachteil ist, dass das Verlassen des Hosts ein Pausieren des Spiels im Zuge einer sogenannten Host-Migration nach sich zieht [7].

D. Netzwerkmodelle in der Praxis

In der Praxis kann oft auch eine Kombination von mehreren Netzwerkmodellen Einsatz finden [3, Chapter 11 – Future Directions]. So wird bei *Battlefield 1*, einem bekannten Online Shooter, das Server-Client-Netzwerkmodell mit autoritativem Game-Server und autoritativen Client-Hosts eingesetzt. Grundsätzlich verwaltet jeder Client für sich das Spielgeschehen. Hat ein Client jedoch eine schlechte Verbindung, so übernimmt der Server die Entscheidungen. Dazu wird eine Ping-Obergrenze festgelegt, bei deren Überschreitung ein Autoritätswechsel erfolgt [7]. Dieses System bringt den Vorteil mit sich, dass Spieler mit einer schlechten Verbindung weiterhin am Spiel teilnehmen können, aber Spieler mit einer guten Verbindung nicht darunter leiden müssen. Ebenso wird der Server entlastet, da er nicht jede Berechnung tätigen muss.

IV. Probleme

Unabhängig vom verwendeten Netzwerkmodell gibt es Probleme mit denen Clients in Netzwerken konfrontiert werden. Die wichtigsten Erscheinungen sind hierbei Paketverlust, Latenzausbrüche und *Jitter*.

A. Paketverlust

Das Verschwinden eines Datenpakets bei der Übertragung wird als Paketverlust bezeichnet. Dies stellt für Echtzeitanwendungen, wozu auch Online Games zählen, ein großes Problem dar. Die für das Spielgeschehen kritischen Updates gehen verloren oder müssen erneut gesendet werden [7]. Durch die auftretende Verzögerung entsteht eine Asynchronität zwischen den Clients, was sich negativ auf die Spielqualität auswirkt. Paketverlust kann durch fehlerhafte Datenpakete, Routingfehler oder Überlastung des Netzwerkes entstehen [3, Chapter 5 – Network Latency, Jitter and Loss]. Zudem tritt Paketverlust häufig bei kabellosen Verbindungen auf. Gründe dafür sind beispielsweise Interferenzen oder schwache Signalstärke. TCP wirkt Paketverlust entgegen, indem es eine Neuübertragung durchführt. Dies wird erreicht, indem jedes Paket bestätigt werden muss. Bleibt im Falle eines verlorenen Pakets eine Bestätigung aus, so wird die Übertragung wiederholt [2, Chapter 13 – Reliable Stream Transport Service (TCP)].

B. Hohe Latenz

Eine hohe Latenz ist von daher kritisch, da durch die Verzögerung der Übertragung, die gesendeten, sowie empfangenen Updates verspätet ankommen. Die Clients treffen Entscheidungen über das Spielgeschehen, ohne die Updates der Clients mit Latenzproblemen miteinzubeziehen. Es kann zu einem inkonsistenten Spielstatus kommen [5]. Als klassisches Beispiel kann man einen beliebigen *Online-Shooter* nehmen. Durch die hohe Latenz sind beide Clients nicht mehr synchron. Ein Spieler schießt auf einen anderen, trifft ihn bei seiner Anzeige des Spielgeschehens. Der andere Spieler ist jedoch auf seiner Anzeige zum Zeitpunkt des eintreffenden Schusses bereits an einem anderen Ort und er wird verfehlt (vgl. Abbildung 4). Hohe Latenzwerte können durch die geografische Distanz zwischen den Computern entstehen. Zudem kann durch Probleme beim Paketrouting der Netzweg auch ohne geografische Distanz sehr lang werden. Je nach Protokoll und Größe der zu sendenden Daten müssen zum Austausch einer Nachricht mehrere Datenpakete versendet werden. Man spricht von mehreren *Round Trips* [2, Chapter 13 – Reliable Stream Transport Service (TCP)]. Beispielsweise muss bei einer Übertragung mittels TCP zuerst eine Verbindung aufgebaut werden. Durch die dafür notwendige Kommunikation entsteht eine zusätzliche Verzögerung bis zum Nachrichtenempfang. Grundsätzlich kann die Latenz nur verbessert werden, indem man eine bessere infrastrukturelle Anbindung an die Spielserver sicherstellt. Dies ist jedoch keine Garantie, dass die Latenz durch Störungen und Ähnliches nicht doch hohe Werte annimmt [12].

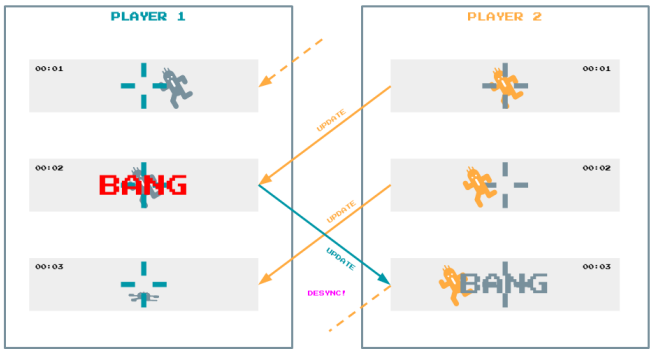

Abbildung 4. Verzögerte Anzeige bei hohem Ping (von [5])

C. Jitter

Jitter zeichnet sich dadurch aus, dass aufeinanderfolgende Datenpakete variierende Latenzen aufweisen. Je höher die Schwankungen werden, desto schwieriger wird es für Clients und Server sich auf ankommende Updates zeitlich einzustellen [3, Chapter 5 – Network Latency, Jitter and Loss]. Konstante Verzögerung durch hohe aber stabile Latenz ist vorhersehbar und kann somit berücksichtigt werden. Wechselt die Verzögerung jedoch in kurzen Intervallen, so kann dies sogar die Paketreihenfolge beeinflussen. Ältere Datenpakete können

vor neueren Datenpaketen ankommen. Jitter ist oft die Folge von Netzüberlastung durch zu viele aktive Geräte. Zudem kann es durch schwache Hardware, also Router, Switches oder sogar Kabel, ausgelöst werden. Jitter tritt ebenso häufiger bei kabellosen Verbindungen auf.

D. Umgang mit Störungen

Durch die nachfolgenden besprochenen Technologien können Verzögerungen bis zu einem gewissen Maße reguliert werden. Jedoch haben auch diese ihre Grenzen, weshalb man in vielen Online Games immer noch eine Obergrenze für Ping und Paketverlust festlegt, die nicht überschritten werden darf. Viele Spiele trennen die Verbindung zu Clients, die einen viel zu hohen Ping haben oder wo die Paketverlustrate ein kritisches Ausmaß annimmt. Oft sind Störungen aber auch stark genug, dass das Spielerlebnis eingeschränkt wird, aber die Obergrenze noch nicht erreicht wird. Solche grenzwertigen Fälle werden auf andere Weise adressiert. Entwicklerstudios setzen hier auf Transparenz [7]. Ein Spieler bekommt über Meldungen mitgeteilt, wenn eine Störung auftritt und was deren Ursache ist. Zudem erhält er die Möglichkeit die Spielperformance auch hinsichtlich Faktoren, wie Paketverlust, Ping, Round Trip Time und ähnlichem zu überwachen (vgl. Abbildung 5). Dies ermöglicht es auftretende Probleme zu analysieren und gegebenenfalls sogar zu beheben [13]. Zudem werden oft auch Einstellungen zur Verfügung gestellt, die Spielern mit schlechten Verbindungen aushelfen. Zum Beispiel ist es möglich auf Kosten von erhöhter Verzögerung ein flüssigeres Spielerlebnis zu erreichen, indem die Frequenz an ausgehender Datenkommunikation reduziert wird [12].

Abbildung 5. Performance Monitoring in Valorant (von [12])

V. Lag Compensation

Da es unmöglich ist die zuvor besprochenen Probleme völlig zu vermeiden, gibt es Technologien, die ihre Auswirkungen mindern. Man bezeichnet diese als *Lag Compensation* [3, Chapter 6 – Latency Compensation Techniques]. Der Sinn dieser Verfahren ist es den Spielern auftretende Verzögerungen zu verschleiern, um ein flüssiges Spielerlebnis aufrechtzuerhalten. Lag kann gemindert werden, indem man die Ansprüche an die Ressourcen reduziert. Beispielsweise wird die Verbindung weniger belastet, wenn die benötigte Bandbreite reduziert wird. Man spricht hierbei von Kompensationstechniken. Dazu zählen Nachrichtenkomprimierung und Aggregation. Durch die Komprimierung von Nachrichten müssen kleinere Datenmengen übertragen werden, was eine Einsparung an Bandbreite ermöglicht. Die Aggregation von Nachrichten dient dazu die Frequenz der Übertragung zu reduzieren, indem der Inhalt mehrerer Nachricht in einer einzelnen versendet wird. Das Bündeln von Nachrichten ermöglicht die Einsparung von Protokollheadern und somit auch Bandbreite [6]. Ein weiterer Aspekt ist das sogenannte Interessenmanagement. Darunter versteht man die Reduktion der übermittelten Nachrichten durch Definition von Interessenbereichen [14]. Dies kann bei Online Games mit großen Spielwelten eingesetzt werden. Die Updates eines Clients werden nur an Clients versandt, die von diesen überhaupt erst beeinflusst werden. Updates werden nur an Spieler im selben Gebiet oder gar in Sichtreichweite versandt. Alle anderen bleiben von der Aktualisierung unbeeinflusst, da diese ohnehin für sie irrelevant wäre.

A. Client Side Predicition

Die wichtigste Maßnahme zur Kompensation von Lag ist die *Client Side Prediction*. Diese Technologie basiert darauf, dass jeder Client für sich die anderen Clients simuliert und deren nächste Schritte vorhersagt. Die Vorhersage erfolgt auf der Historie von getätigten Kommandos [11]. Es wird also basierend auf den vergangenen Aktionen die wahrscheinlichste nächste Aktion bestimmt. Die Clients führen nun selbst Logik aus und berechnen das Spielgeschehen [10]. Der Server verwaltet jedoch weiterhin den Spielstatus. Bei einem autoritativen Server muss dieser die Simulationen bestätigen oder korrigierend eingreifen. Je besser der Algorithmus zur Vorhersage, desto niedriger ist die Anzahl an nötigen Korrekturen durch den Server. Ein weiteres Merkmal für einen guten Algorithmus ist es, wenn die Korrekturen unauffällig erfolgen können. Bei der Korrektur von Vorhersagen spricht man von *Server Reconcilation* [15]. Betrachtet man den zeitlichen Ablauf der Kommunikation, dann befindet sich der Client in der Gegenwart. Der Server verarbeitet dessen Nachrichten zu einem späteren Zeitpunkt. Der Client erhält die Antwort vom Server nochmals später. Somit beziehen sich seine Antworten, beziehungsweise Korrekturen auf die Vergangenheit. Der Client ist zum Zeitpunkt des Empfangs einer Antwort bereits einige Schritte weiter [16].

Würde ein Client die Korrektur direkt umsetzen, so würden sich Charaktere in einem Online Game plötzlich an andere

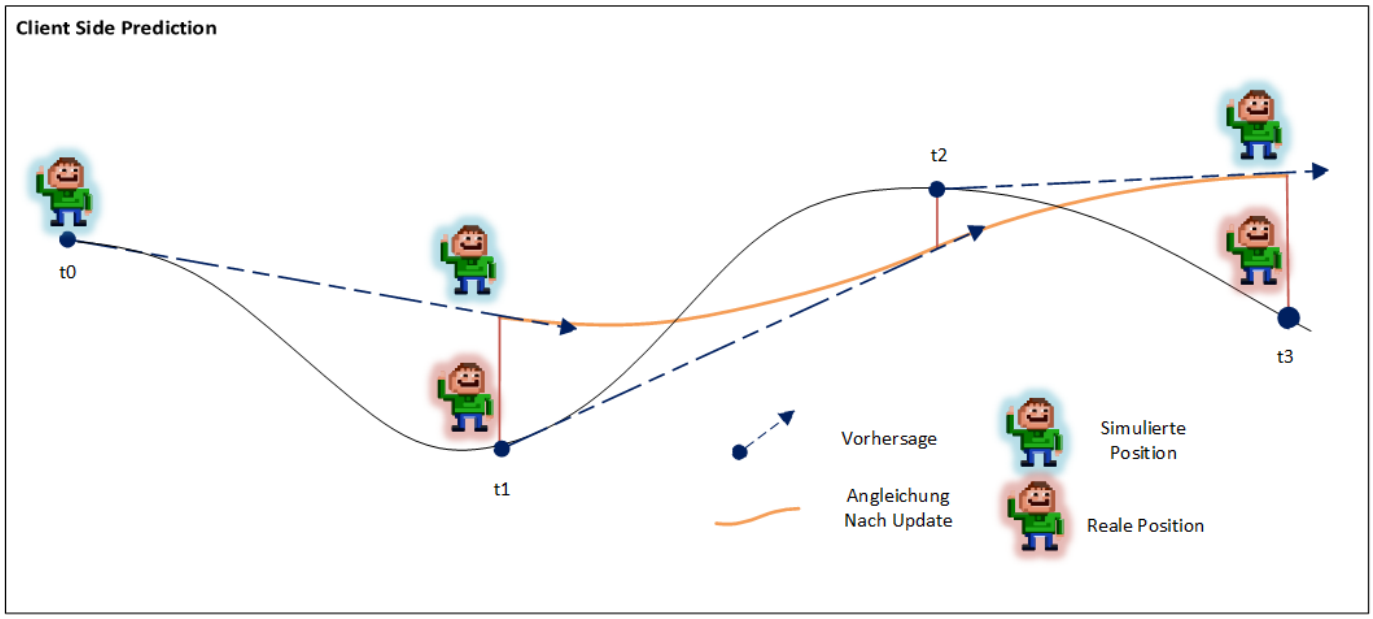

Abbildung 6. Client Side Predicition mit sanfter Korrektur

Orte teleportieren, ungeachtet von der zuvor erfolgten Vorhersage [10]. Die Lösung für dieses Problem ist das *Data Buffering*. Man verwendet einen Ringspeicher, der den Status der Clients beinhaltet [3, Chapter 6 – Latency Compensation Techniques]. Bei einer auftretenden Korrektur wird der ältere Inhalt des Speichers geleert. Alle anderen Einträge werden so geändert, dass sie zum korrekten Zustand passen. Die Angleichung an das wirkliche Spielgeschehen erfolgt somit nicht plötzlich, sondern schrittweise [17]. Anhand Abbildung 6 kann nachvollzogen werden, wie der Client auf ein Update vom Server reagiert. Die simulierte Position wird aufseiten des Clients dargestellt. Vom Server bekommt der Client in regelmäßigen Zeitabständen die reale Position des Charakters mitgeteilt. Zwischen diesen Zeitpunkten muss er die Charakterposition möglichst akkurat simulieren. Erhält der Client zu einem Zeitpunkt die Information über die Position eines Charakters und stellt fest, dass diese von seiner Berechnung abweicht, so muss eine Angleichung erfolgen. Dazu wird von der realen Position aus eine Vorhersage über den Weg getroffen, den der Charakter zurücklegen wird. Anschließend wird von der simulierten Position aus zum Endpunkt der Vorhersage ein Weg berechnet. Dadurch erfolgt keine abrupte Korrektur, sondern eine sanfte Anpassung bis zum Zeitpunkt der nächsten Aktualisierung durch den Server. Heutzutage schicken Server solche Aktualisierungen 60 Mal die Sekunde an alle Clients [13]. Dementsprechend müssen Vorhersagen nur für Bruchteile einer Sekunde erfolgen und Korrekturen sind gleichermaßen mikroskopisch.

B. Simulationsdivergenz

Eine große Abweichung zwischen Realität und Simulation sollte bei Spielen vermieden werden. Gehen wir beispielsweise von einem Online-Shooter aus. Ein Spieler gibt einen Schuss ab auf seinen Gegner kurz bevor dieser seine Deckung erreicht. Auf seiner Anzeige hat der Schuss getroffen. Auf der Gegenseite jedoch zeigt das Spiel dem anderen Spieler an, dass er die Deckung bereits erreicht hat. Somit kann der Schuss ihn nicht treffen. Der Server muss jetzt entscheiden, was der richtige Spielstatus ist. Durch die stark abweichenden Simulationen wird einer der Spieler jedoch diese Entscheidung anzweifeln. Solche Szenarien sollen unter allen Umständen vermieden werden. Dazu muss die Simulation auf beiden Seiten möglichst identisch sein. Eine Simulation auf Seite der Clients erfolgt grundsätzlich zwischen zwei *Server-Ticks* [7]. Bei einem Server-Tick handelt es sich um den Zeitpunkt, zu dem die Simulation berechnet und das Ergebnis an alle Clients versendet wird. Wenn man vom Begriff *Tickrate* spricht, ist darunter die Anzahl an Updates pro Sekunde zu verstehen, welche meist in Hertz angegeben wird. Online Shooter wie *Team Fortress 2* haben beispielsweise eine Tickrate von 66 Hz [13]. Das heißt ein Server berechnet 66 Mal die Sekunde die Spielsimulation neu und informiert alle Clients. Ein erster Schritt, um einer Simulationsdivergenz entgegenzuwirken ist der Einsatz von Servern mit hohen Tickraten. Je mehr Server-Ticks durchgeführt werden, desto geringere Zeitabstände müssen durch die Clients vorhergesagt werden. Jedoch benötigt ein Server auch zunehmend Rechenleistung, da er viel schneller und öfters das Spielgeschehen simulieren muss. Gleichermaßen steigt die Netzlast durch die erhöhte Kommunikation mit den Clients. Dies kann sich negativ auf die Stabilität des Servers auswirken. Die Steigerung der Tickrate des Servers ist somit nur zu einem gewissen Maße möglich. Eine weitere Verbesserung muss auf Seite der Clients erfolgen.

1) Interpolation: Ein wesentlicher technischer Aspekt, um Divergenz von Realität und Simulation zu vermeiden, ist die

Interpolation. Interpolation ist eine Technik zur Ermittlung von Übergängen zwischen zwei bekannten Werten. Ein Client interpoliert also zwischen Updates von zwei aufeinanderfolgenden Server-Ticks. Dazu wird die Anzeige auf Seite der Clients in die Vergangenheit gesetzt. Bei der *Source Engine*, die unter anderem bei *Counter-Strike* Einsatz findet, ist die Anzeige auf den Clients beispielsweise um 100ms verzögert [11]. Der Vorteil ist, dass die Clients dadurch gleichzeitig mehrere Updates vom Server vorliegen haben. Es müssen lediglich Zwischenwerte zwischen zwei realen Werten ermittelt werden. Selbst wenn mehr als ein Datenpaket verloren geht, kann eine Interpolation durchgeführt werden. Der Abstand zwischen den vorliegenden Werten ist dann größer, was zu einer stärkeren Divergenz führen kann [18, Chapter 6: Entity Interpolation and Prediction]. Es wird eine sehr flüssige Anzeige ermöglicht trotz Störungen. Hierbei werden keine Vorhersagen getroffen, da nur mit Werten vom Server gerechnet wird. Dadurch ist die Anzeige sehr realitätsnah. Die zusätzliche Verzögerung wird dann durch den Server kompensiert. Bei mäßiger künstlicher Verzögerung merkt der Spieler nicht, dass er eigentlich die Vergangenheit sieht.

Die einfachste Form der Interpolation ist die lineare Interpolation. *Valorant*, ein Online-Shooter aus dem Jahr 2020, nutzt zur Interpolation den linearen Ansatz [12]. Wie der Name vermuten lässt, wird hier ein Zwischenwert zwischen zwei Werten abgeleitet, indem von einem linearen Übergang zwischen Anfang und Ende ausgegangen wird. Stellen wir diesen Sachverhalt in einer Formel dar, so ergibt sich die interpolierte Position wie folgt:

$$position = start \cdot (1 - time) + end \cdot time$$

Hierbei stellen *start* und *end* jeweils die Anfangs- und Endposition dar. *time* ist ein Wert zwischen 0 und 1. Der Wert gibt an zu welchem Zeitpunkt in der Transition man sich befindet, wobei 0 der Ausgangspunkt und 1 das Ziel darstellt. Setzt man für *time* 0 in die Formel ein, so ergibt sich:

$$position = start \cdot (1 - 0) + end \cdot 0 = start$$

Gleichermaßen erhält man für *time* gleich 1:

$$position = start \cdot (1 - 1) + end \cdot 1 = end$$

Für beliebige Werte von *time* erhält man also die Position, die zu dem jeweiligen Zeitpunkt erreicht wurde. Dadurch kann man mit zwei gegebenen Werten beliebige Zwischenwerte interpolieren [19].

In der Praxis speichert sich ein Client ankommende Aktualisierungen zu Positionen in einem Puffer. Jede Aktualisierung wird mit dem Zeitpunkt ihrer Ankunft hinterlegt. Kommen neue Aktualisierungen an, so können ältere entfernt werden. In Listing 1 wird die Implementierung einer simplen Interpolation dargestellt. Man versetzt den Client eine Zeiteinheit in die Vergangenheit, um eine Interpolation auszuführen. Man liest zwei Zeitpunkte aus dem Puffer, die vor und nach dem Moment in der Vergangenheit liegen. Nun wird zwischen den Positionen, die ein Charakter zu diesen Zeitpunkten hatte, interpoliert [18].

```
1  void Update() {
2    if (networkView.isMine) return; // don't run
       interpolation on the local object
3    if (stateCount == 0) return; // no states to
       interpolate
4    double currentTime = Network.time;
5    double interpolationTime = currentTime -
       InterpolationBackTime;
6    // the latest packet is newer than
       interpolation time - we have enough packets
       to interpolate
7    if (stateBuffer[0].Timestamp >
       interpolationTime) {
8      for (int i = 0; i < stateCount; i++) {
9        // find the closest state that matches
       network time or use oldest state
10       if (stateBuffer[i].Timestamp <=
       interpolationTime || i == stateCount - 1) {
11         // the state closest to network time
12         networkState lhs = stateBuffer[i];
13         // the state one slot newer
14         networkState rhs =
       stateBuffer[Mathf.Max(i - 1, 0)];
15         // use time between lhs and rhs to
       interpolate
16         double length = rhs.Timestamp -
       lhs.Timestamp;
17         float t = 0 f;
18         if (length > 0.0001) {
19           t = (float)((interpolationTime -
       lhs.Timestamp) / length);
20         }
21         transform.position =
       Vector3.Lerp(lhs.Position, rhs.Position, t);
22         break;
23       }
24     }
25   }
26 }
```

Listing 1. Verbesserte Interpolation(von [18])

Neben der einfachen linearen Interpolation gibt es weitere Verfahren, die bessere Ergebnisse liefern. Dazu zählen beispielsweise Polynominterpolation oder Spline-Interpolation [20, Chapter 5 Interpolation]. Diese Verfahren können mehr Informationen verarbeiten und erzielen so höhere Genauigkeit. Die Implementierung des Algorithmus ist komplizierter. Der Grundaufbau bleibt jedoch gleich.

Wichtig bei der Interpolation ist, dass die Clients nicht zu weit in die Vergangenheit gesetzt werden. Man muss Interpolation also vorsichtig verwenden. Bei zu intensiver Nutzung der Technik wirkt es sich negativ auf das Spielerlebnis aus, da die Latenz erhöht wird. Die serverseitige Lag Compensation kann nur bis zu einem gewissen Maß Verzögerungen ausgleichen.

2) Extrapolation: Ein weiterer Aspekt zur Vermeidung von Simulationsdivergenz ist die Extrapolation. Diese Technik basiert auf dem Prinzip des *Dead Reckoning*. Hierbei wird angenommen, dass Änderungen bezüglich der Bewegung selten vorkommen. Änderungen von Kurs und Geschwindigkeit sind verhältnismäßig selten. Man kann sich also an vergangenen Updates orientieren, um Vorhersagen zu treffen [16]. Extrapolation ermöglicht die näherungsweise Abbildung von verzögerten oder ausbleibenden Paketen. Bei Ankunft der echten Daten können diese dann korrigierend eingefügt werden. Im Gegensatz zur Interpolation wird keine künstliche

Verzögerung benötigt. Jedoch erreicht dieses Verfahren eine schlechtere Genauigkeit der Simulation [21].

Moderner Netcode nutzt beide Verfahren. Zur Simulation zwischen Server-Ticks nutzen die Clients Interpolation. Der Server setzt Extrapolation ein, um die Zukunft der Clients vorherzusagen. Er sendet diese Vorhersagen als Updates an alle Clients. Aufgrund der natürlichen Netzwerkverzögerung kommen die extrapolierten Werte zu dem Zeitpunkt an, wo sie nicht mehr die Zukunft, sondern die Gegenwart darstellen. Die Clients haben also nicht mehr nur vergangene Server-Updates vorliegen [16] [17]. Dadurch kann die für die Interpolation notwendige künstliche Verzögerung umgangen oder zumindest reduziert werden.

VI. Zusammenfassung und Ausblick

Betrachten wir alle zuvor genannten Abhängigkeiten und stellen die Frage was guten Netcode ausmacht. Es ist essenziell ein passendes Netzwerkmodell zu verwenden, sowie leistungsstarke Hardware einzusetzen, um hohe Tick-Raten auf Serverseite zu ermöglichen. Ebenfalls muss eine niedrige Latenz gewährleistet werden. Dies kann zum Beispiel durch eine gute Infrastruktur oder das Festlegen von Obergrenzen für die Clients erzielt werden. Lag Compensation ist ein wichtiger Bestandteil, der ein reaktionsschnelles Spielen ermöglicht. Insbesondere große Unternehmen entwickeln hier ausgefeilte Algorithmen, um sowohl den Spielern mit hohen Pings als auch den Spielern mit einer stabilen Internetverbindung ein gleichermaßen gutes Spielgefühl zu ermöglichen. Zusammenfassend muss jedoch betont werden, dass trotz aller moderner Technik ein Minimum an Bandbreite und Stabilität der Verbindung für das Online Gaming unumgänglich ist.

Der Themenbereich, der sich unter dem Buzzword Netcode eröffnet, wurde durch diesen Bericht erläutert. Die grundlegenden Prinzipien von Networking wurden erklärt. Ebenso wurden die verschiedenen Aspekte von Netcode dargestellt. Dazu zählt die Netzwerkarchitektur, also die typischen Netzwerkmodelle. Eine ebenso große Rolle spielt die Server-Logik. Hier finden Technologien und Algorithmen zur Lag Compensation ihren Einsatz. Ein weiterer wichtiger Aspekt ist die Sicherheit, insbesondere der Einsatz von *Anti-Cheat-Software*. Dieser Teilbereich des Networking ist jedoch so umfangreich, dass er den Rahmen des Berichts sprengen würde und wurde deshalb nur am Rande angesprochen.

Mit dem gewonnenen Wissen ist es möglich Phänomene wie beispielsweise *Rubber banding* technisch zu erklären. Unter Rubber banding versteht man ein durch Latenz verursachtes stockendes Teleportieren eines Charakters in einem Online Game. Ebenso kann man die Algorithmen zur Lag Compensation tiefgehend betrachten. Insbesondere ist die Implementierung von solchen Algorithmen interessant und stellt ein mögliches Thema für eine zukünftige Arbeit dar. Eine Arbeit zum Thema *Network-Security* von Online Games wäre gleichermaßen vorstellbar. Hier könnte auf Erkennung, Abwehr und Prävention, sowie die unterschiedlichen Möglichkeiten eines Angriffs eingegangen werden.

Literatur

[1] R. Immoos, "Die Gaming-Industrie – grösser als die Hollywood- und Musikbranche." Online Article, Mar. 2020, Accessed: 29.11.2020. [Online]. Available: https://www.hwzdigital.ch/die-gaming-industrie-groesser-als-die-hollywood-und-musikbranche/#:~:text=Ein%20durchschnittliches%20Wachstum%20von%2012,2019%20Global%20Games%20Market%20Report).

[2] D. Comer, *Internetworking with TCP/IP*. Upper Saddle River, N.J: Prentice Hall, 2000.

[3] G. Armitage, *Networking and online games : understanding and engineering multiplayer Internet games*. Chichester, England Hoboken, NJ: John Wiley & Sons, 2006.

[4] D. Lowe, *Networking all-in-one desk reference for dummies*. Hoboken, N.J. Chichester: Wiley, 2005.

[5] Y. G. (Meseta), "Netcode concepts part 1-3," Online Article, Apr. 2018, Accessed: 12.01.2020. [Online]. Available: https://medium.com/@meseta/netcode-concepts-part-1-introduction-ec5763fe458c

[6] J. Smed, T. Kaukoranta, and H. Hakonen, "Aspects of networking in multiplayer computer games," *The Electronic Library*, vol. 20, no. 2, pp. 87–97, apr 2002.

[7] C. "Battle(non)sense", "How netcode works, and what makes 'good' netcode," Online Article, Oct. 2017, Accessed: 12.01.2021. [Online]. Available: https://www.pcgamer.com/netcode-explained/

[8] R. Alam and M. Tariq, "A survey on wired and wireless network," 04 2019.

[9] /r/leagueoflegends, "[poll] what is your average ping? (seperate polls for na, euw. eune)," Online Poll, May 2015, Accessed: 02.01.2021. [Online]. Available: https://www.reddit.com/r/leagueoflegends/comments/369lol/poll_what_is_your_average_ping_seperate_polls_for/

[10] G. Fiedler, "What every programmer needs to know about game networking," Online Article, Feb. 2010, Accessed: 02.01.2021. [Online]. Available: https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/

[11] Y. W. Bernier, "Latency compensating methods in client/server in-game protocol design and optimization," Online Article, May 2017, Accessed: 12.01.2021. [Online]. Available: https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization

[12] D. S. Matt deWet, "Peeking into valorant's netcode," Online Article, Jul. 2020, Accessed: 18.02.2021. [Online]. Available: https://technology.riotgames.com/news/peeking-valorants-netcode

[13] T. Edwards, "Source multiplayer networking," Online Article, Apr. 2019, Accessed: 12.01.2021. [Online]. Available: https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

[14] J. Smed, *A review on networking and multiplayer computer games*. Turku: Turku Centre for Computer Science, 2002.

[15] G. Gambetta, "Fast-paced multiplayer part 1-4," Online Article, Jul. 2017, Accessed: 12.01.2021. [Online]. Available: https://www.gabrielgambetta.com/client-server-game-architecture.html

[16] C. Savery and T. C. N. Graham, "Timelines: simplifying the programming of lag compensation for the next generation of networked games," *Multimedia Systems*, vol. 19, no. 3, pp. 271–287, jun 2012.

[17] A. Pushak, "Client-side prediction for smooth multiplayer gameplay," Online Article, May 2017, Accessed: 12.01.2021. [Online]. Available: https://www.kinematicsoup.com/news/2017/5/30/multiplayerprediction

[18] A. Stagner, *Unity multiplayer games : build engaging, fully functional, multiplayer games with Unity engine*. Birmingham, UK: Packt Pub, 2013.

[19] webdva, "How i implemented client-side linear interpolation," Online Article, Jan. 2019, Accessed: 12.01.2021. [Online]. Available: https://webdva.github.io/how-i-implemented-client-side-linear-interpolation/

[20] J. R. Chasnov, "Numerical methods," Free Online Text, Feb 2021, Accessed: 18.02.2021. [Online]. Available: https://www.math.ust.hk/~machas/numerical-methods.pdf

[21] doc.photonengine.com, "Photon bolt," Online Documentation, Jan. 2020, Accessed: 12.01.2021. [Online]. Available: https://doc.photonengine.com/en-us/bolt/current/in-depth/interpolation-vs-extrapolation