Mobiledevice Controller Framework

Ron Schiwkowksi (915074), Hannes Grothknopf (915449), Michael Schleiss (923739), Dennis Hofmann (919285), Christian Heinrichs (919020)

University of Applied Sciences, Sokratesplatz 1, 24149 Kiel, Germany

ABSTRACT

Aktuelle Browser bieten auf Basis von HTML5 die Möglichkeit zur Echtzeitkommunikation per Websocket und auf mobilen Endgeräten das Auslesen von Sensordaten. Beide Techniken sollen eingesetzt werden, um ein kabelloses Gamepad mit vielfachen Möglichkeiten zur Steuerung eines Spiels, zum Beispiel über die Rotation des mobilen Endgeräts oder dem Touchscreen, zu entwickeln.

Keywords: Mobile, Smartphone, JavaScript, HTML5, Nodejs, API

1. EINLEITUNG

Das vorliegende Dokument beschreibt die Entwicklung eines Frameworks, welches es erlaubt verschiedenste Anwendungen im öffentlichen Raum zu installieren, die mit Hilfe von mobilen Controllern gesteuert werden können. Hierbei stehen zwei Aspekte im Vordergrund. Zum einen soll das Framework eine Schnittstelle bieten, eigene Anwendungen unkompliziert in eine stabile Infrastruktur einzubinden. Dies bedeutet, dass alle notwendigen Funktionen der Kommunikation und der Nutzerverwaltung durch das Framework abgedeckt sind und der Entwickler unabhängig davon flexibel eine eigene Anwendung realisieren kann. Zum anderen soll das Framework dem Endbenutzer die einfache Bedienung der Anwendung über persönliche mobile Geräte - wie Smartphones - ermöglichen. Hierbei sollen möglichst alle Arten der Eingabe des Endgerätes genutzt und dieses so in ein multifunktionales Gamepad verwandelt werden. Im Folgenden wird nach einer Übersicht der verwendeten Technologien zunächst die zugrunde liegende Struktur des Servers und der dort implementierten Funktionen zum Client- bzw. User-Management erläutert. Anschließend werden die Controller für die Verwendung auf dem Smartphone vorgestellt. Hier wird besonders auf die Verwendung der zu Verfügung stehenden Sensoren und Ausgabemöglichkeiten auf mobilen Geräten eingegangen. Zuletzt werden die Ergebnisse eines Lasttests ausgewertet und geben so einen guten Einblick in die Performance des Systems.

1.1 Problemstellung

Die Möglichkeit ein Spiel komfortabel mit einem mobilen Gerät zu verbinden und zu steuern soll im folgenden unseren Beschreibungen zugrunde liegen. Hierzu werden verschiedene Technologien miteinander verbunden und genutzt.
Eine Möglichkeit zu schaffen, komfortabel an einem Spiel teilzunehmen, als Beispiel an einem Flughafen per QR-Code einem laufenden Spiel beizutreten, motivierte uns zu diesem Thema.

2. METHODIK

2.1 Projektmanagement

Das Team besteht aus 5 Projektmitgliedern. Die Verteilung der Aufgaben ist in Tabelle 2.1.1 aufgestellt. Verschiedene Aufgaben sind im Gitlab erfasst und Personen zugewiesen. Ein wöchentliches persönliches Treffen aller Projektmitglieder führt zu einer guten Arbeitsmoral und hohen Effektivität.

2.1.1 GitLab

Das Projekt wurde auf einem eigenen GitLab gehostet und den Projektmitgliedern zur Verfügung gestellt. Die Issues des Systems wurden genutzt um Aufgaben zu verteilen und zu dokumentieren.
Strukturiert wurde das Arbeiten nach Feature Branches, um das Arbeiten am Projekt möglichst flexibel zu halten. Der Master-Branch diente als Release-Branch, in den regelmäßig die Änderungen per Merge Request eingetragen und getestet wurden.

Tabelle 1. Matrix der Aufgabenverteilung

Aufgabe	Schleiss	Grothkopf	Schiwkowski	Hofmann	Heinrichs
Recherche	X	X	X	X	X
Konzept	X	X	X	X	X
Serverdesign		X			X
Networking		X		X	X
GameAPI		X			X
DemoGame	X				X
ClientFrontend	X		X		
ClientBackend	X		X		
UserMgmnt				X	
Dokumentation	X	X	X	X	X
Projektmgmt.					X

2.2 Technologien

NodeJS / Javascript
NodeJS ist eine Laufzeitumgebung für JavaScript und basiert auf der Chrome's V8 JavaScript engine. NodeJS dient als Kern unserer Applikation und stellt alle Dienste bereit.

HTML 5
In diesem Projekt setzen wir auf HTML5, um die Sensordaten der Geräte auszulesen. Dies ist durch das neue HTML5 einfach umzusetzen.

2.2.1 Javascript Bibliotheken

jQuery
jQuery ist eine schnelle, umfangreiche und einfach zu bedienende JavaScript-Bibliothek. Sie ermöglicht Dinge wie das einfache modifizieren von HTML, Ereignisverarbeitung, Animation und Ajax.

require.js
RequireJS ist eine Bibliothek welche dazu dient, einfach JavaScript-Dateien zu modularisieren und zu laden. Es ist für den In-Browser-Einsatz optimiert, kann allerdings auch in anderen JavaScript-Umgebungen wie Rhino und Note verwendet werden.

socket.io
Das Framework „socket.io“ wird in Abschitt 3.1.1 ausführlich beschrieben.

MongoDB
MongoDB ist eine Schema-freie, dokumentenorientierte Open-Source-Datenbank, die in der Programmiersprache C++ geschrieben ist. Da die Datenbank dokumentenorientiert ist, kann sie Sammlungen von JSON-ähnlichen Dokumenten verwalten. Dadurch lässt sie sich leicht mit JavaScript zusammen benutzen und wurde von uns gewählt.

2.2.2 Hardware

Die Server Hardware kann je nach Bedarf eine Server- oder Desktopumgebung sein. Der NodeJS Server kann plattformunabhängig auf so gut wie allen Plattformen genutzt werden.
Als Eingabegerät sind Mobile Geräte wie Smartphones oder Tablets angedacht. Mit ihren Sensoren und Touchdisplays sind diese prädestiniert.

3. IMPLEMENTIERUNG

3.1 Architektur

Network
Die Netzwerkverbindungen bilden einen wichtigen Bestandteil der Anwendung. Alle Clients und das Server-Frontend bzw. Spiel verbinden sich zum Server via WebSocket.

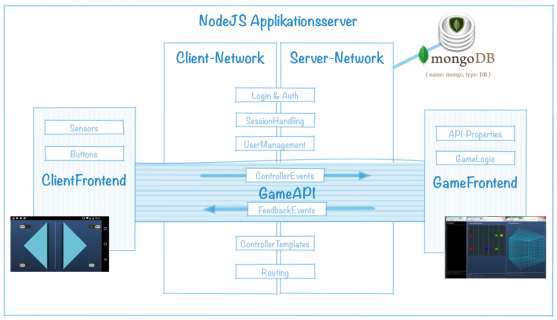

Abbildung 1. Architektur der Applikation

3.1.1 socket.io

Zum Aufbau und Handling der Netzwerkverbindungen wird die Bibliothek socket.io von Guillermo Rauch verwendet. Sie kapselt die WebSocket API Spezifikationen und bietet eine vereinfachte und einfach zu benutzende API zu den WebSockets auf Client und Server-Seite. Außerdem verfügt sie über Failover-Strategien, die das Verwenden von anderen Protokollen vorsehen, für den Fall, dass WebSockets vom Client oder Server nicht unterstützt werden.
Auf node Server Seite werden die Pakete 'http' und 'socket.io' benötigt. Erstes um einen HTTP-Server zu erstellen und letzteres, um diesen dann zu einer WebSocket Verbindung aufzuwerten und diese dann zu verwalten. Grundlegend wird die Verbindung mit Hilfe von message Events implementiert, welche auf dem jeweiligen Socket ausgelöst und abgehört werden können. Dieses Prinzip ähnelt stark den von node bekannten Events. Eine Client-Version von socket.io wird vom Server automatisch veröffentlich und kann so auf dem Client direkt eingebunden und verwendet werden.*

3.1.2 Server Network Handling

Die Anwendung besteht aus einem zentralen node Server, einem Frontend zur Darstellung des Spiels und einer unbestimmten Anzahl an mobilen Clients. Alle Komponenten stellen jeweils eine WebSocket Verbindung zum Server her, der diese so zentral verwalten kann. Durch die Implementation der selbst geschriebenen Module 'gameApi' und 'clientNetwork' werden die Funktionen der WebSockets erneut gekapselt und ermöglichen eine stark angepasste Verwendung.
Der Server hält zu jeder Zeit eine Liste aller offenen Socket-Verbindungen, was es ermöglich individuell mit jedem Client und dem Frontend zu kommunizieren und eingehende Nachrichten direkt einem Absender zuzuordnen. Auf diesem Level der Kommunikation wurden grundlegende Socket-Events zum Verbindungsaufbau, der Nutzerverwaltung sowie dem Verschicken der eigentlichen Nachrichten implementiert. Abhängig vom Typ der eingehenden Nachricht werden die Funktionen der nächst-höheren Programmschicht aufgerufen, die diese dem Netzwerk Modul bei der Initialisierung per Callback übergeben hat. Die eigentlichen Socket-Events werden an dieser Stelle nicht weitergegeben, um die technische Socket-Logik komplett im Modul zu kapseln. Die Struktur der einzelnen Nachrichtentypen wurde genau definiert. Besonders im Fall der tatsächlichen Nachtrichten Pakete wurde ein weiteres 'type' Attribut eingefügt, die es dem Spieleentwickler erlaubt, unabhängig von den Socket Events eigene Nachrichten-Typen zu definieren. Ein weiterer wichtiger Aspekt ist die Art und Weise, in der die Sockets auf Client-Seite aufgebaut werden.

*https://davidwalsh.name/websocket [3] [4]

Zusätzlich bietet das 'serverNetwork' Modul eine Reihe an Funktionen an, mit den Clients und dem Frontend zu kommunizieren und diese zu verwalten.[†]

3.2 Session

Die nächste Schicht nach dem Netzwerk stellt auf Server-Seite das Modul 'sessionHandling' dar. Hier werden die noch anonymen Socket-Verbindungen bzw. deren IDs mit tatsächlichen Nutzernamen und der zugrundeliegenden Nutzerverwaltung verknüpft. Dies bedeutet zunächst, dass jeder Client sich zum Benutzen der Anwendung am Server authentifizieren muss. Jede eingehende Login-Anfrage wird an das Modul 'UserManagement' weitergeleitet um die übergebenden Nutzerdaten zu verifizieren. Nach einem erfolgreichen Login werden die Daten des Nutzers in eine Liste aktiver Nutzer gespeichert und sein Nutzername wird an die nächste Schicht weiter gegeben. Beim Disconnect eines Nutzers wird sein Eintrag aus der Liste gelöscht und es wird ebenfalls eine Nachricht an die nächste Programm-Schicht übergeben. Alle Nachrichten, die tatsächliche Kommunikationsdaten beinhalten, werden direkt weitergegeben.
Ein Login bedeutet notwendigerweise eine zuvor erfolgte Registrierung, welche ebenfalls vom 'UserManagement' Modul verwaltet wird. Um die Registrierung jedoch nicht zu einer unnötigen Barriere in der Verwendung der Applikation zu machen, ist es ebenfalls möglich sich anonym einzuloggen. In diesem Fall wird der jeweiligen Socket-Verbindung ein zufälliger Nutzername zugewiesen. Der Vorteil eines Nutzerkontos bietet die Möglichkeit für das Spiel, Session übergreifend Daten für die Benutzer zu speichern.
Die Login-Sessions sind direkt an eine bestehende Socket-Verbindung gekoppelt und werden beendet, sobald der Socket geschlossen wird.

3.3 User Management

Das „UserManagement“ kümmert sich um die Verwaltung der Nutzer. Dafür stellt es Funktionen für die Registrierung, Authentifizierung sowie für das setzen und holen von Benutzerdaten zur Verfügung. Die Funktionen im „UserManagement“ bekommen eine callback Funktion übergeben, welche bei einer erfolgreichen Ausführung ausgeführt wird.

3.4 Datenbank

Das "Database"Modul kümmert sich um die persistente Speicherung aller anfallenden Daten. Dafür wird eine MongoDB verwendete und die für das hinzufügen, löschen, updaten und für Abfragen benötigten Funktionen zur Verfügung gestellt.

3.5 Mobile - Sensoren und Daten

Die Basis aller weiteren Schritte ist das Auslesen der Sensordaten eines geeigneten Gerätes, während der gesamten Entwicklung wurden hierfür Smartphones unterschiedlicher Hersteller und Generationen verwenden. Die Daten wurden auf dem Gerät gemessen, aufbereitet und als gekapselte Datenpakete an den vorgesehenen Empfänger gesandt. Die Datenaufbereitung ist ein wichtiger Schritt, um die Netzwerklast am Knotenpunkt, dem Server, möglichst gering zu halten, da mehrere Geräte gleichzeitig eine Verbindung herstellen und aufrecht erhalten können und dies zu einer Überlast des Empfängers bei unkontrolliertem Sendevolumen führt.
Die von der Applikation genutzten Daten sind im folgenden erläutert.

Touchbildschirm Die übliche Art der Interaktion mit einem aktuellen Smartphone, die Toucheingabe, darf natürlich nicht außer Acht gelassen werden. Da diese in der Entwicklung äußerst einfach zu implementieren ist, unteranderem über JQuerys .tap(), kann diese Eingabe als trivial angesehen und hier nicht weiter darauf eingegangen werden.

[†] http://stackoverflow.com/questions/7709289/how-to-pass-javascript-object-from-one-page-to-other und http://tavendo.com/blog/post/websocket-persistent-connections/

Rotations- und Beschleunigungssensor
Diese Werte bilden den Kern der Applikation und sollen hier näher dargestellt werden:
Die Interaktion über die Bewegung des Smartphones im dreidimensionalen Raum ist intuitiv und gut für jede denkbare Aufgabe geeignet. Die Datenerhebung erfolgt dabei für gewöhnlich durch Differentialkondensatoren und Drei-Achsen-Gyroskope. Da die Daten vom Gerät kontinuierlich in festgelegten Intervallen bestimmt werden, können diese auf komfortable Art und Weise ausgelesen werden. Die aufwändigere Arbeit ist die anschließende Verarbeitung und Filterung der Daten. Dies geschieht über ein zu setzendes Event, namentlich das „devicemotion" -Event, das bei jeder Änderung der Rotation oder Beschleunigung feuert und seine Daten beispielhaft in der Form event.acceleration.x; oder event.accelerationRate.alpha; bereitstellt, ebenso wie das Intervall der gemessenen Daten (in Millisekunden) über event.interval; verwendet werden kann. Siehe auch Anhang F10.
Die Originalversion der Rotation und Berechnungen sind zu finden in den Aufzeichnungen von Damian Gordon[‡].
Die Aufbereitung der Daten geschieht über eine Filterung aller leeren und ungültigen Daten, außerdem aller Daten, die darauf schließen lassen, dass keine Bewegung stattfand. Um eine Überlast der Datenverbindung zu verhindern wird eine Mittelung der ausgelesenen Daten als adäquates Mittel angesehen, diese kann über eine festgelegte Anzahl der Werte vorgenommen werden. Da Smartphones allerdings in einem unterschiedlichen Intervall Daten Messen, in der Entwicklung wurde eine Messung im Abstand von 50ms, 100ms und 200ms festgestellt, kann die Mittelung auch über einen gewissen Zeitwert vorgenommen werden.

Ausrichtung des Gerätes Die Orientierung des Gerätes wird als „Portrait" oder „Landscape", also vertikale oder horizontale Ausrichtung beschrieben. Diese Daten verwenden dieselbe Technik wie der Beschleunigungssensor und wird beispielsweise in der Bildschirmausrichtung verwendet. So einfach wie das Prinzip ist auch die Verwendung: Die Ausrichtung lässt sich über window.orientation; (in Grad) ausmessen. Ein Wert über 180 beschreibt dabei den Portraitmodus, über 180 den Landscape- oder Landschaftsmodus.

Bild und Tonaufzeichungen
Da dies einer der komplexeren Datentypen ist, den man mit dem Handy aufzeichnen kann, empfiehlt sich für die Aufnahme eine leicht benutzbare API. Mit getUserMedia() ist für Bild und Ton nun eine einfache Alternative verfügbar, die Stückweise in die aktuelle Browsergeneration eingefügt wird. Die aktuelle Verfügbarkeit innerhalb der Browser kann auf caniuse.com eingesehen werden und ergibt das Bild wie in Anhang E9 aufgeziegt.

Vibration Im Gegensatz zu den anderen Daten geht es hierbei nicht um eine Ein-, sondern um eine Ausgabemethode. Die Benachrichtigung über Vibrationen ist in JavaScript denkbar einfach gestaltet, ist aber, ebenso wie die vorherigen APIs, nicht in jedem Browser verfügbar. Der letztliche Aufruf der Funktion geschieht über den Navigator, explizit über window.navigator.vibrate(), wobei die Länge und Art der Vibration angegeben werden kann.

Browserempfehlung Da (noch) nicht alle Browser alle APIs unterstützen, soll hier eine Empfehlung zur Verwendung der oben genannten Funktionalitäten genannt werden. Sowohl auf dem Desktop-Rechner als auch unter Android ist derzeit die aktuelle Version von Firefox (43) am 15.12.2015 erschienen und in der Lage, alle genannten Funktionen und APIs auszuführen.

3.6 Graphical User Interface and JavaScript

Den Kern der grafischen Oberfläche bilden zwei Pfeile, die nach dem Prinzip eines Steuerkreuzes funktionieren und dazu genutzt werden, die Geschehnisse auf dem verbundenen Rechner zu kontrollieren. Die Benutzeroberfläche wurde absichtlich übersichtlich und simpel gehalten, um eine schnelle Akzeptanz zu gewährleisten. Aus Test- und Präsentationsgründen wurde in jeder Ecke des Templates eine Checkbox platziert, die jeweils eine Art von Sensordaten steuert. Speziell die gezielte Deaktivierung und Aktivierung der Audio- und Videodaten wurde als wichtiges Kriterium angesehen, da dies einen gefährlichen Eingriff in die Privatsphäre darstellen kann. Aus diesem

[‡] http://de.slideshare.net/DamianGordon1/3d-robot-movements

Grund ist eine Aktivierung der genannten Multimediadaten nicht ohne vorherige Bestätigung möglich.
Jeder Tastendruck wird vom dahinterliegenden JavaScript aufgefangen und direkt an den Rechner weitergeleitet, wo er entsprechende Aktionen auslöst. Um den Benutzer in der Bedienung Sicherheit zu geben, wird für jeden Tastendruck ein optisches Feedback gegeben, entweder durch direkte Aktionen auf dem Rechner oder als Hinweis auf dem Smartphone selbst.

3.7 Nutzen der GameAPI

Der Entwickler kopiert das in Anhang A kommentierte Code-Skelett in sein Spiel und füllt die gekennzeichneten Stellen mit seiner Spielsteuerung.

4. DEMONSTRATION

4.1 Vorbereitung

Ein Computer stellt den NodeJS Server.
Ein weiterer Computer stellt das gewählte Demogame-Frontend auf einem großen Bildschirm dar.
Mehrere Clients verbinden sich mit ihren mobilen Geräten auf den Server.
Anschließend müssen die vom NodeJS Server benötigten Abhgängikeiten installiert werden. Dies geschieht mit einen Aufruf des Skriptes „prepareDeps.sh“.

4.2 Start der Applikation

Der NodeJS Server wird mit dem Befehl wie in Listing 1 gestartet. Imfolgenden sollte eine Meldung erscheinen welche den Applikationsserver als gestartet meldet. Eine Beispielhafte Meldung in in Listing 1 zu sehen.

```
1 node NodeServer/bin/www
  16 Jan 14:39:22 serverNetwork | Server listening on port 5222
```

Listing 1. Start der Applikation

4.3 Nutzen einer Demo Applikation

Controller Interface
Die Clients rufen die Adresse des Servers auf und es wird ihnen eine Login Anzeige dargestellt. Hier wird nun eine Entscheidung getroffen, ob ein Login erstellt, genutzt oder anonym weiter fortgefahren werden soll. Wird ein Login genutzt, wird anstelle eines generierten Namens der tatsächliche Benutzername im gewählten Demogame angezeigt.

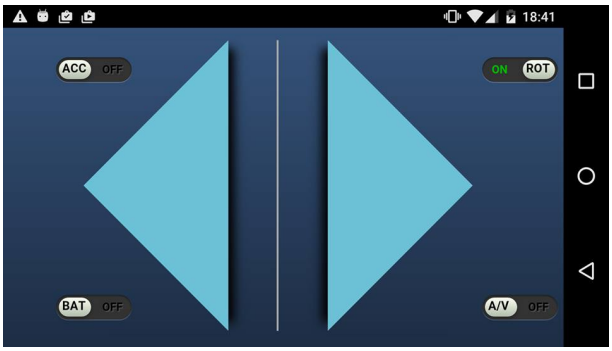

Abbildung 2. Unterstützung der getUserMedia API auf caniuse.com

Demo Applikation
Auf einem Browser, welcher als Frontend dient, muss nun das entsprechende Game über die URL aufgerufen werden. In der Demo-Applikation ist dies „/game/ 1—2—3 /“ Die nun verbundenen Clients führen mit ihren Geräten unterschiedliche Aktionen durch. Ein Druck auf einen der Buttons auf einem der Clientgeräte wird als solcher identifiziert und im Frontend dargestellt. Ein anderer Client aktiviert die Rotationserfassung und die gesendeten Daten erscheinen auf dem Frontend.

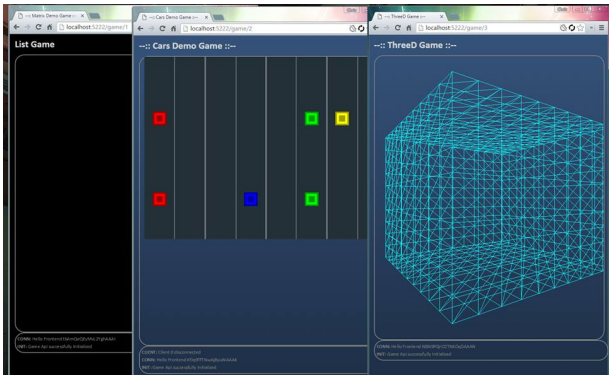

Abbildung 3. Unterstützung der getUserMedia API auf caniuse.com

5. LASTTESTMESSUNG

5.1 Server

Szenario Der Server wird nach und nach mit immer mehr Paketen belastet und die CPU-Auslastung ermittelt.

Durchführung
Mithilfe des node-inspectors[§] wurde die NodeJS-Serveranwendung analysiert. Der node-inspector ermöglicht eine Nutzung der Entwicklertools des Chrom-Browsers. Zur Ermittlung der CPU-Last wurden Profile bei verschiedenen Lastzuständen erstellt, welche 30 Sekunden lang aufrecht erhalten wurden. Das Testsystem war dabei mit einem Intel(R) Xeon(R) CPU E3-1241 v3 @ 3.50GHz ausgestattet.

Ergebnisse
Anhand der Ergebnisse ist gut der lineare Anstieg der Auslastung zu erkennen. Das Diagramm 4 stellt die Ergebnisse von 0 bis 5500 Paketen dar. Auf dem Diagramm 4 ist gut zu erkennen, dass die Auslastung ab circa 3500 Paketen in einem geringeren Maße ansteigt, allerdings ist auch hier ein linearer Verlauf zu erkennen. Die Auslastung schwankt zwar je nach eingesetztem System, der lineare Anstieg wird sich jedoch übertragen lassen. Um dies zu bestätigen sind weitere Messungen auf verschiedenen Testsystemen nötig.

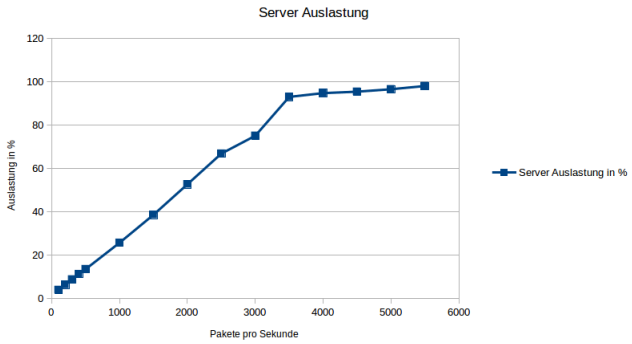

Abbildung 4. Serverlast bei 0 - 5500 Paketen

5.1.1 Frontend

Test Szenario Zur Messung der Latenz wird ein Gerät mit dem Server verbunden und jede Sekunde dessen durchschnittliche Latenz vom Senden der Daten auf dem Gerät bis zur Ausgabe dieser durch die API erfasst. Die Auslastung des Servers wird durch das Senden von zusätzlichen Paketen erreicht. Die Senderate wird nach und nach gesteigert, um die maximale Auslastung der API auf dem Testsystem zu ermitteln. Zu beachten ist, dass auf allen Geräten vergleichbare Zeitstempel vergeben werden, um die Ergebnisse nicht zu verfälschen. Die Tests werden dabei ohne die Anbindung einer Anwendung an die API durchgeführt. Zu beachten ist, dass diese auch Leistung benötigt, was die Ergebnisse drastisch verändern kann. Aufgrund des Fokus auf die Entwicklung einer API ohne eine gezielte Anwendung wurde diese nicht berücksichtigt.

[§]https://github.com/node-inspector/node-inspector

Durchführung Zur Erfassung der Daten wurde die Anzahl der gesendeten Pakete und die durchschnittlichen Latenz pro Sekunde von einem Gerät ermittelt. Während der Messung wurde der Server mit einer unterschiedlich starken Anzahl an zusätzliche Paketen belastet um eine höhere Auslastung zu erreichen. Die Werte wurden dabei in einem Abstand von einer Sekunde ermittelt. Zum Erfassen der Daten wurden die Chrome-Entwicklertools verwendet. Es wurden für jede Laststufe Daten über 50 Sekunden erfasst. Der Server und das Frontend liefen bei dem Test auf dem gleichen System.

Ergebnisse
Die Ergebnisse der Messung sind in dem Diagramm 5 zu erkennen. Der Mittelwert gibt die durchschnittliche Latenz über 50 Sekunden an. Durch die mittlere Abweichung ist gut zu erkennen, dass es zu gelegentlichen Latenzspitzen kommt, welche im Betrieb unter Umständen zu Problemen führen können. Ab einer Paketanzahl von 5000 ist ein drastischer Anstieg der durchschnittlichen Latenz zu erkennen. Unter der Annahme, dass ein Gerät 100 Pakete die Sekunde versenden kann, kann im Betrieb ohne Anwendung eine auf dem Testsystem maximal mögliche Anzahl von 50 Geräten angenommen werden. Mit Blick auf die Daten von der Serverleistungsmessung in Abbildung 4 sollte die Anzahl von 35 Geräten nicht überschritten werden und der Bereich von 35-50 Als Pufferzone eingeplant werden.

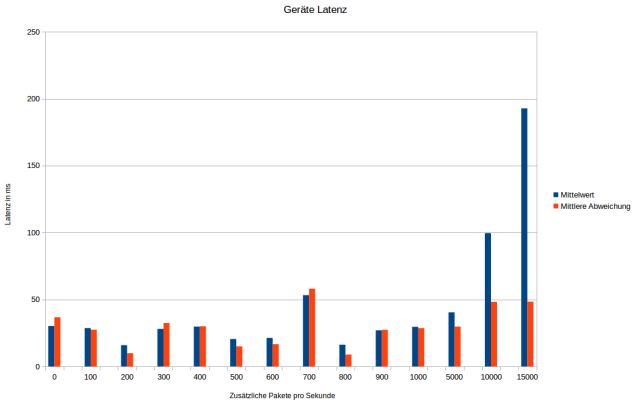

Abbildung 5. Geräte Latenz Messung

6. AUSBLICK

Glättung der Sensorwerte Ein noch zu lösendes Problem ist die Glättung der von den Sensoren kommenden Werte. Je nach Gerät liefern die Sensoren unterschiedlich stark schwankende Daten. So feuerten manche Testgeräte, selbst auf dem Tisch liegend, permanent Orientierungsdaten, obwohl es nicht bewegt wurde. Diese Schwankungen müssen erkannt und unterbunden werden. Ein Threshhold ist bereits implementiert und sollte perspektivisch genau abstimmt werden.

Begrenzung der gesendeten Sensordaten Um den Server nicht ungleich durch die verschiedenen Geräte zu belasten und die Auslastung berechenbar zu machen, muss das Sendeverhalten reglementiert werden. Die optimalen Werte dafür müssen noch durch weitere Leistungstest ermittelt werden und hängen auch von der Anwendung ab, welche an die API angebunden ist.

7. FAZIT

Die Umsetzung zeigt ein flexibles und belastbares Framework, welches es ermöglicht ein Spiel unkompliziert mit einem mobilen Gerät zu bedienen. Ein Spieleentwickler hat die Möglichkeit eigene Controller-Templates bereitzustellen und mühelos sein Spiel zu integrieren. Dazu nutzt er den Beispielcode und bedient sich der Eingabeevents des Frameworks, welche generiert werden. Das Usermanagement ermöglicht zudem eine komfortable Individualisierung der Spieler.

APPENDIX A.

```
/** gameApi.controllerTemplates.*
* Hier traegt der Entwickler ein, welches Gamepad-Controller Layout
* auf dem Smartphone des Spielers angezeigt werden soll
* */
gameApi.controller = gameApi.controllerTemplates.MODERN;
/**
 * HANDSHAKE - bitte nicht modifizieren
 * */
gameApi.frontendConnection = function (connInfoObj) {
    gameApi.addLogMessage(gameApi.log.INFO, 'conn', connInfoObj + " " + gameApi.
        socket.id);
    this.emit('frontendOutboundMessage', {type: 'setControllerTemplate', data:
        gameApi.controller});
};

/**
 * Handling der eingehenden Daten
 * Hier wird die Steuerung des Spiels mit den Steuerungsdaten vom Smartphone
     kombiniert
 * */
gameApi.frontendInboundMessage = function (data) {
    var controllerEvent = controllerData.data.message;
    var clientName = data.data.clientName;
    var message = data.data.message;

    switch (data.type) {
        // Ein Client betritt oder verlaesst das Spiel
        case "userConnection":
            gameApi.addLogMessage(gameApi.log.INFO, 'client', 'Client ' +
                clientName + ' ' + message);

            if (message === 'connected') {
                // Was soll passieren wenn sich ein Client ins Spiel einloggt
            } else if (message === 'disconnected') {
                // Was soll passieren, wenn sich ein Client das Spiel verlaesst
            }
            break;
        // Ein Client sendet ein Button Event
        case "button":
            if (controllerEvent.buttonName == 'btn-left' && controllerEvent.
                buttonState === gameApi.BUTTON.DOWN) {
                // Was soll passieren, wenn Button A gedrueckt wird
            } else if (controllerEvent.buttonName == 'btn-right' &&
                controllerEvent.buttonState === gameApi.BUTTON.DOWN) {
                // Was soll passieren, wenn Button B gedrueckt wird
            }
            break;
        case "accelerationData":
            //Was passiert mit den Beschleunigungsdaten?
            break;
        case "orientationData":
            //Was passiert mit den Rotationsdaten?
            break;
        default:
            break;
    }
```

Listing 2. Nutzen der GampApi

APPENDIX B.

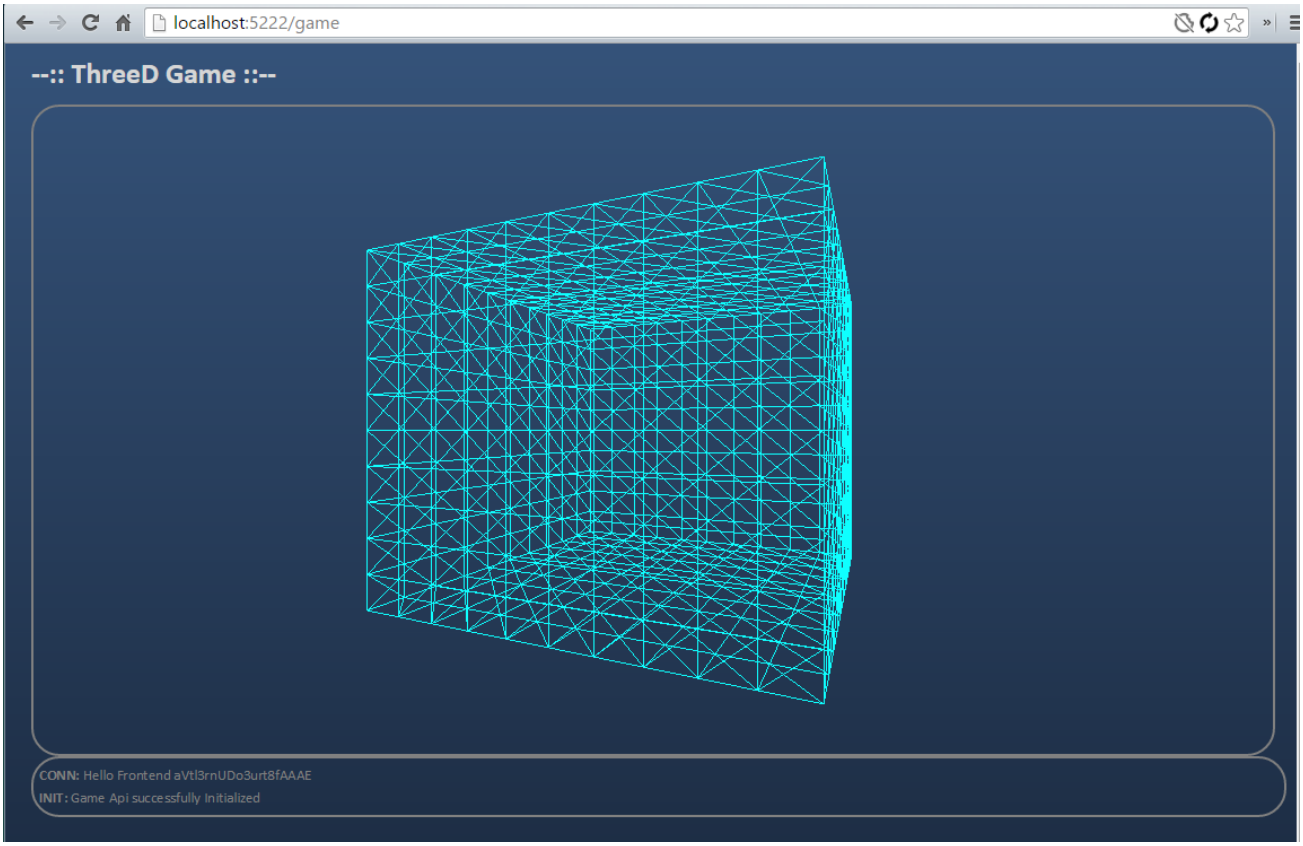

Abbildung B6. 3D-Cube

APPENDIX C.

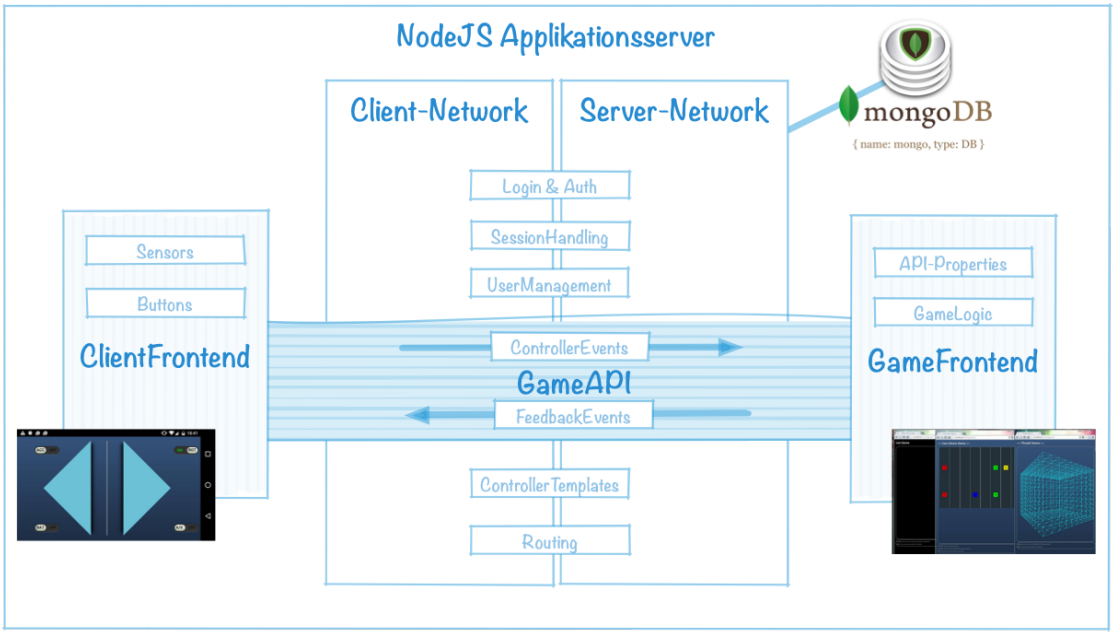

Abbildung C7. m113

APPENDIX D.

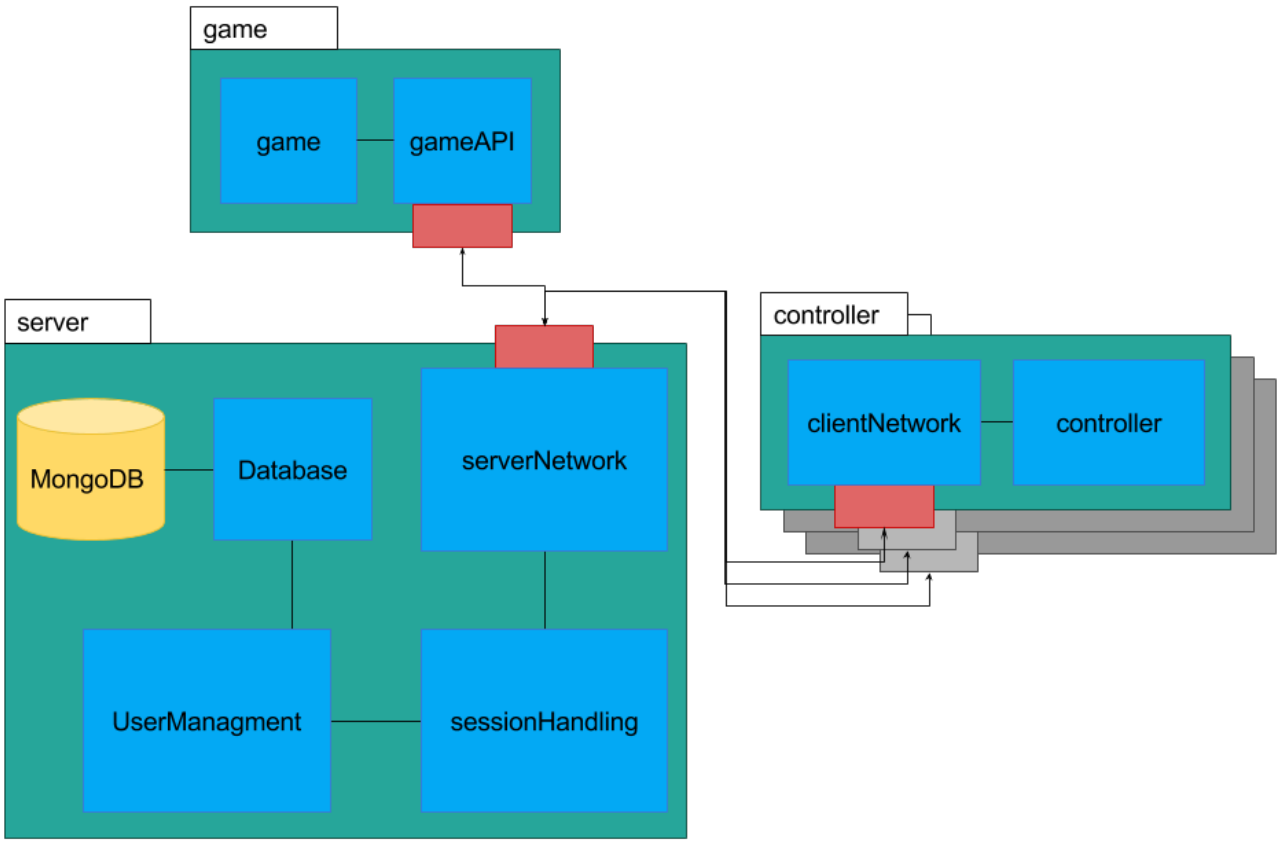

Abbildung D8. Komponentendiagramm

APPENDIX E.

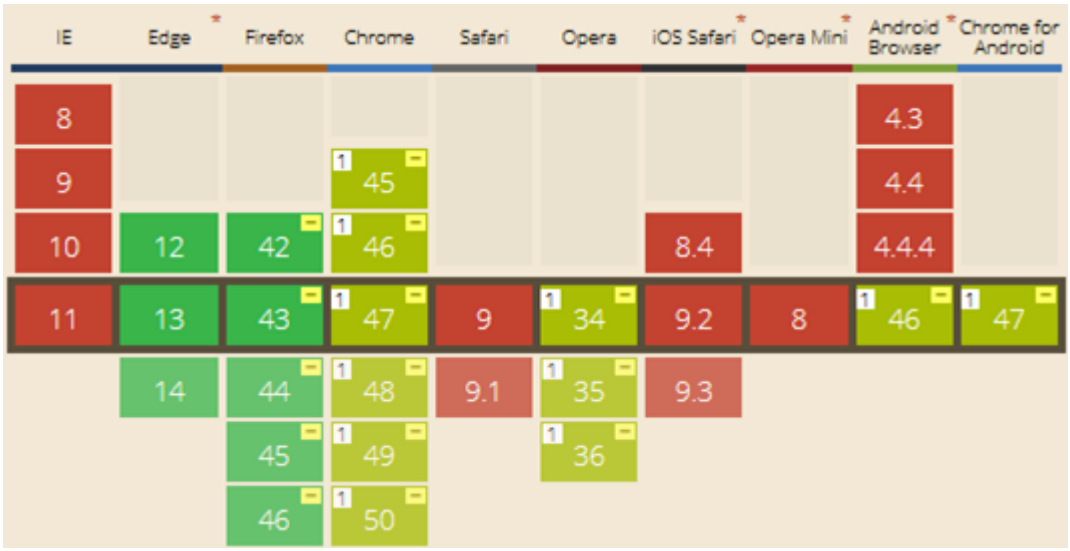

Abbildung E9. Kompatibilität getUserMedia(), caniuse.com

APPENDIX F.

Rotation θ Z

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation θ X

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

Rotation θ Y

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

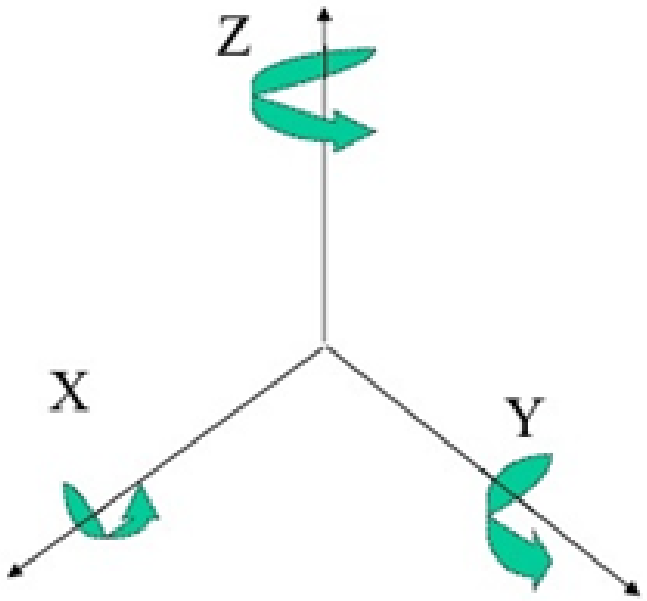

Abbildung F10. Rotationsachsen dargestellt

APPENDIX G.

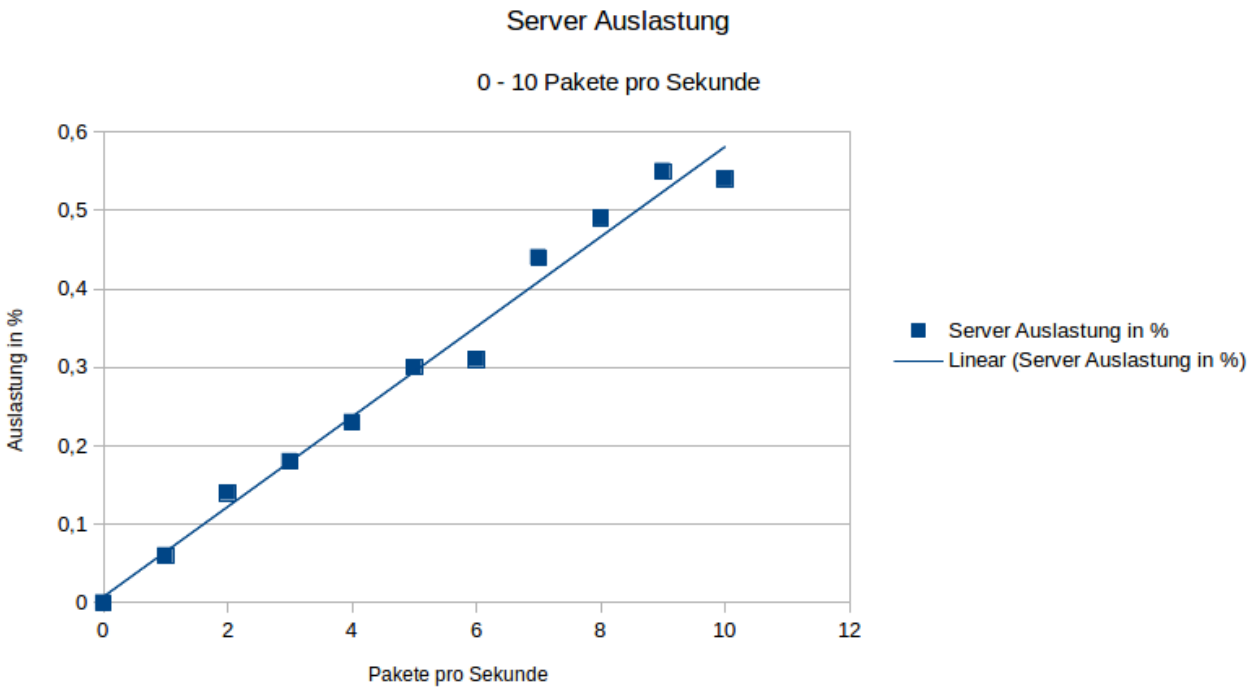

Abbildung G11. ServerLast 0 - 10 Pakete

APPENDIX H.

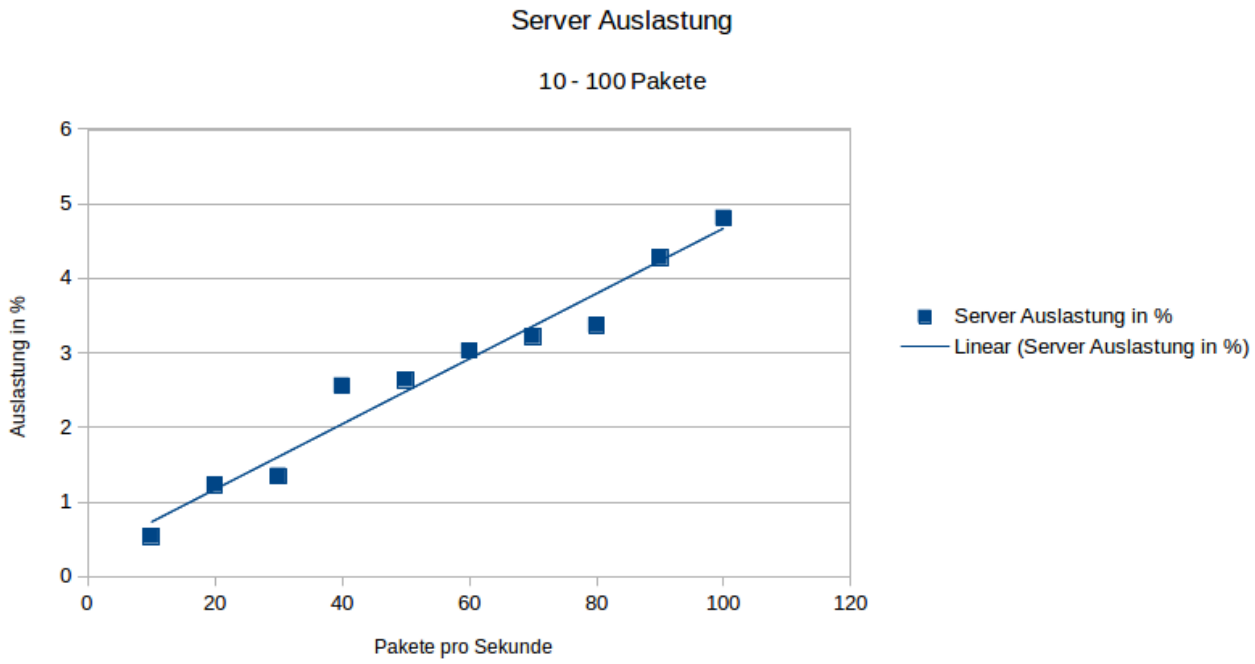

Abbildung H12. ServerLast 10 - 100 Pakete

APPENDIX I.

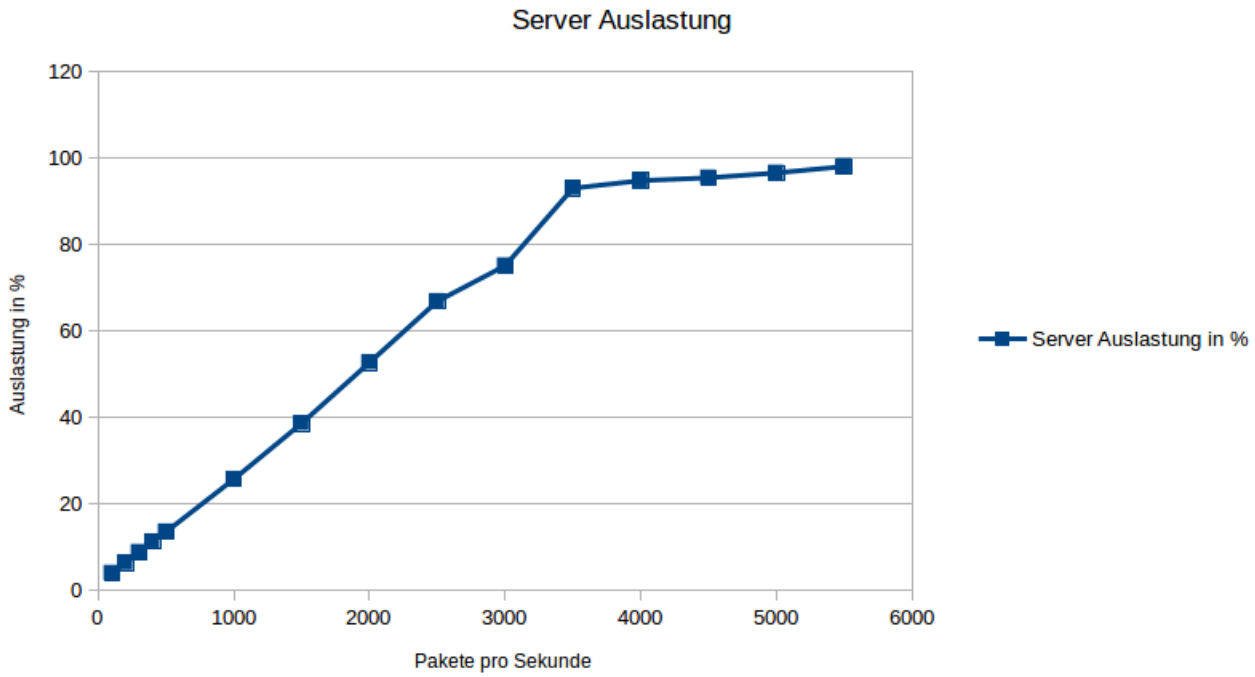

Abbildung I13. ServerLast 0 - 55000 Pakete

APPENDIX J.

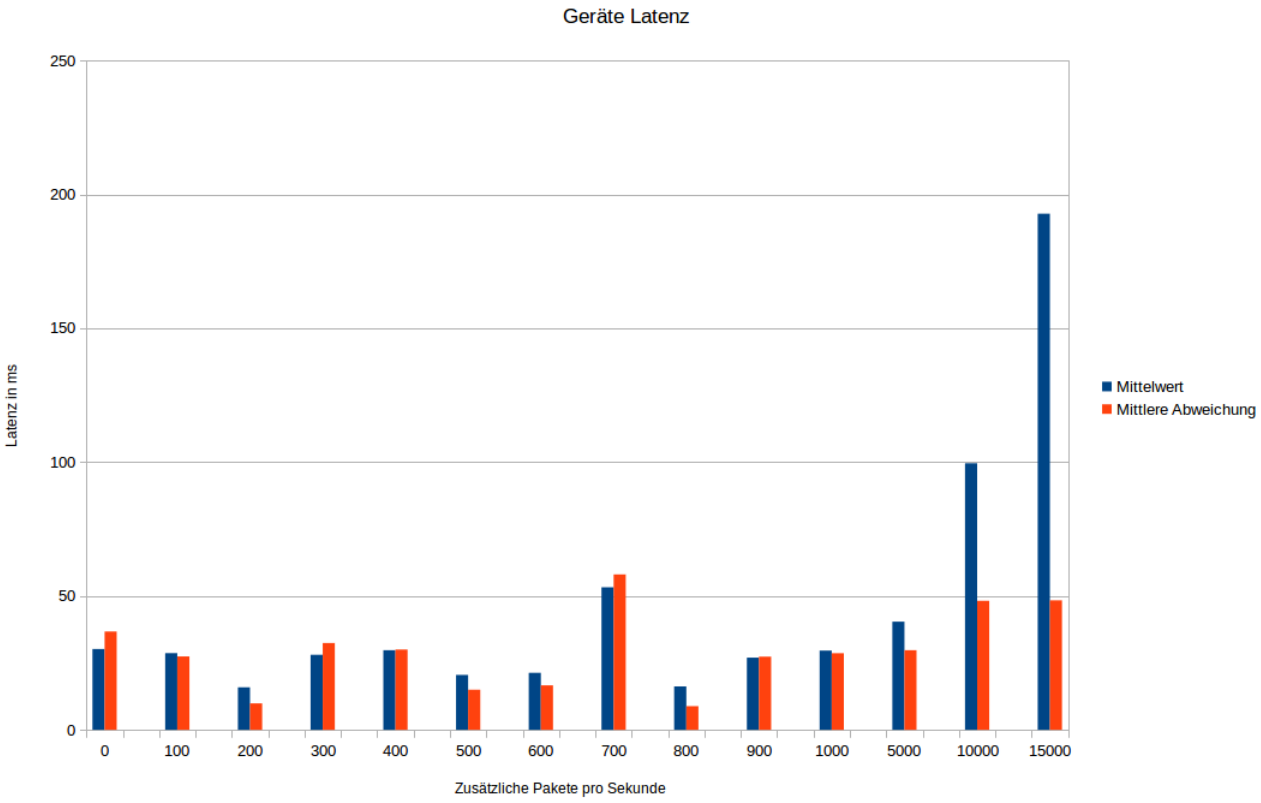

Abbildung J14. Latenzmessung