

Netcode-Mechaniken in Echtzeitanwendungen:  
Analyse von Client-Side Prediction,  
Interpolation und Lag Compensation am  
Beispiel einer Unity-basierten Zielumgebung

Alexander Seitz

14.05.2025

# Contents

<b>1</b>	<b>Einleitung</b>	<b>3</b>
1.1	Motivation . . . . .	4
1.2	Zielsetzung der Arbeit . . . . .	4
1.3	Aufbau der Arbeit . . . . .	4
<b>2</b>	<b>Theoretische Grundlagen</b>	<b>4</b>
2.1	Netzwerke in Echtzeitanwendungen . . . . .	4
2.1.1	Latenz, Paketverlust, Jitter, Tickrate . . . . .	4
2.2	Architekturen in Multiplayer-Systemen . . . . .	4
2.2.1	Client-Server-Modell . . . . .	4
2.2.2	Peer-to-Peer-Kommunikation . . . . .	4
2.2.3	Hybride Topologien . . . . .	4
2.2.4	Authoritätsmodelle . . . . .	4
2.3	Netzwerktheorie . . . . .	4
2.3.1	Tickrate und Input Delay . . . . .	4
2.3.2	Reconciliation, Prediction, Interpolation . . . . .	4
2.4	Mathematische Grundlagen . . . . .	4
2.4.1	Vektoren, Transformationen und Rotationen . . . . .	4
2.4.2	Quaternionen und ihre Bedeutung in 3D-Rotation . . . . .	4
2.4.3	Interpolationsverfahren . . . . .	4
<b>3</b>	<b>C# Beispiel: Unity Netcode</b>	<b>5</b>
<b>4</b>	<b>Literaturverzeichnis</b>	<b>6</b>

# 1 Einleitung

Moderne Echtzeitanwendungen, insbesondere im Bereich der Computerspiele, stellen hohe Anforderungen an die zugrunde liegende Netzwerkarchitektur. Die Synchronisation von Spielzuständen, die Minimierung von Latenzzeiten sowie ein robuster Umgang mit Paketverlust und Jitter sind entscheidend für eine positive Nutzererfahrung. Multiplayer-Systeme, wie sie etwa mit Unity entwickelt werden, müssen deshalb nicht nur funktional, sondern auch leistungsfähig und fehlertolerant sein.

Die vorliegende Arbeit beschäftigt sich mit den theoretischen und technischen Grundlagen der Netzwerkkommunikation in Mehrspielerumgebungen. Anhand eines prototypischen Projekts in Unity werden exemplarisch zentrale Mechanismen wie Reconciliation, Interpolation sowie die Kommunikation im Client-Server-Modell umgesetzt und analysiert.

Im Folgenden werden zunächst die Motivation und Zielsetzung der Arbeit erläutert. Anschließend wird der strukturelle Aufbau der Arbeit vorgestellt.

### 1.1 Motivation

### 1.2 Zielsetzung der Arbeit

### 1.3 Aufbau der Arbeit

## 2 Theoretische Grundlagen

### 2.1 Netzwerke in Echtzeitanwendungen

#### 2.1.1 Latenz, Paketverlust, Jitter, Tickrate

### 2.2 Architekturen in Multiplayer-Systemen

#### 2.2.1 Client-Server-Modell

#### 2.2.2 Peer-to-Peer-Kommunikation

#### 2.2.3 Hybride Topologien

#### 2.2.4 Autoritätsmodelle

### 2.3 Netzwerktheorie

#### 2.3.1 Tickrate und Input Delay

#### 2.3.2 Reconciliation, Prediction, Interpolation

### 2.4 Mathematische Grundlagen

#### 2.4.1 Vektoren, Transformationen und Rotationen

#### 2.4.2 Quaternionen und ihre Bedeutung in 3D-Rotation

#### 2.4.3 Interpolationsverfahren

### 3 C# Beispiel: Unity Netcode

Das folgende Beispiel zeigt, wie man eine einfache `NetworkManager`-Klasse in Unity erstellen kann, die bei der Verbindung von Server und Client hilft. Moderne Echtzeitanwendungen stellen hohe Anforderungen an die Netzwerkarchitektur [1].

```
1 using Unity;
2 using Unity.Netcode;
3
4 public class SimpleNetworkManager : NetworkBehaviour
5 {
6     public override void OnNetworkSpawn()
7     {
8         base.OnNetworkSpawn();
9
10        //kek
11        if (IsServer )
12        {
13            Debug.Log("Server gestartet");
14        }
15        else
16        {
17            Debug.Log("Client verbunden");
18        }
19    }
20
21    [ServerRpc]
22    public void SendMessageServerRpc(string message)
23    {
24        Debug.Log("Nachricht vom Client empfangen: " + message);
25    }
26
27
28    [ClientRpc]
29    public void SendMessageClientRpc(string message)
30    {
31        Debug.Log("Nachricht vom Server empfangen: " + message);
32    }
33 }
```

Listing 1: Unity Netcode Example

## 4 Literaturverzeichnis

### References

- [1] Vercel. *What is Next.js?* n.d. URL: <https://nextjs.org/docs#what-is-nextjs> (visited on 01/16/2025).