universität innsbruck

Increasing throughput of server applications by using asynchronous techniques

A case study on CoAP.NET

Philip Wille
Betreuer: Michael Felderer, Andreas Danek

Programmierparadigmen

- Synchron

Programmierparadigmen

- Synchron

- Asynchron

Programmierparadigmen

- Synchron
  - **Stoppt** den Programmfluss.
- Asynchron

Programmierparadigmen

- Synchron
  - **Stoppt** den Programmfluss.
- Asynchron
  - Kann im Programmfluss weiter gehen.

Programmierparadigmen

- Synchron
  - **Stoppt** den Programmfluss.
  - Überprüft periodisch, ob Funktion beendet ist.
- Asynchron
  - Kann im Programmfluss weiter gehen.

Programmierparadigmen

- Synchron
  - **Stoppt** den Programmfluss.
  - Überprüft periodisch, ob Funktion beendet ist.
- Asynchron
  - Kann im Programmfluss weiter gehen.
  - Ein Event markiert die Beendigung der Funktion.

Programmierparadigmen

- Synchron
  - **Stoppt** den Programmfluss.
  - Überprüft periodisch, ob Funktion beendet ist.
  - Ist als „Blocked“ oder „Waiting“ markiert.
- Asynchron
  - Kann im Programmfluss weiter gehen.
  - Ein Event markiert die Beendigung der Funktion.

Programmierparadigmen

- Synchron
  - **Stoppt** den Programmfluss.
  - Überprüft periodisch, ob Funktion beendet ist.
  - Ist als „Blocked“ oder „Waiting“ markiert.
- Asynchron
  - Kann im Programmfluss weiter gehen.
  - Ein Event markiert die Beendigung der Funktion.
  - Ist frei für andere Aufgaben.

Synchroner Server

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

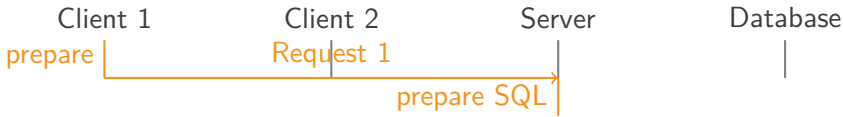

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

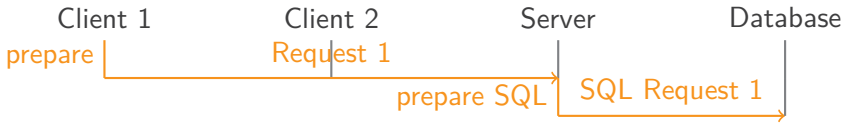

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

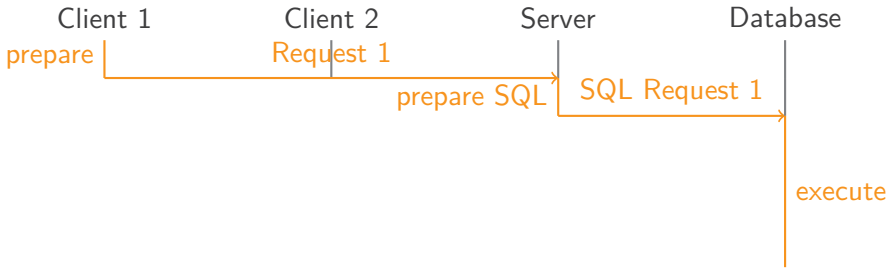

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

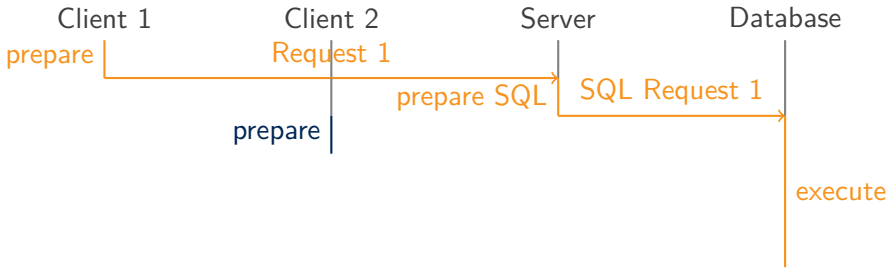

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

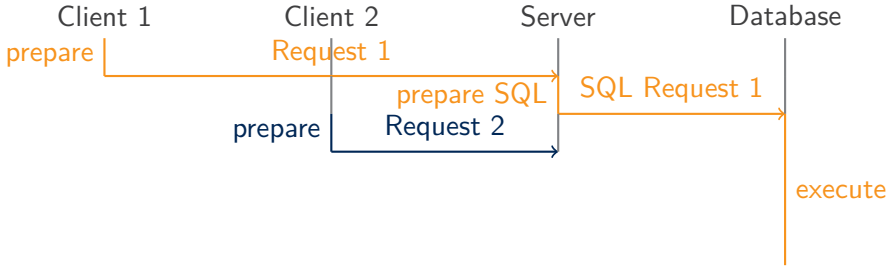

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

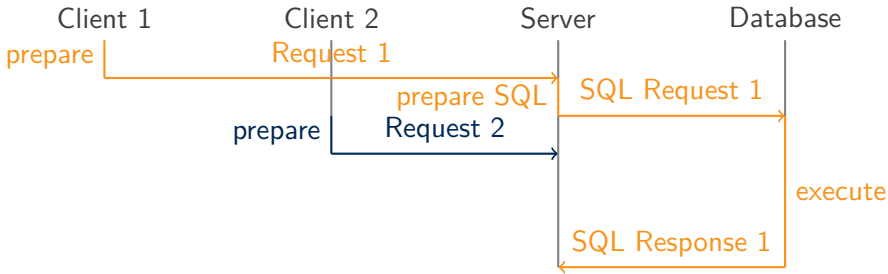

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

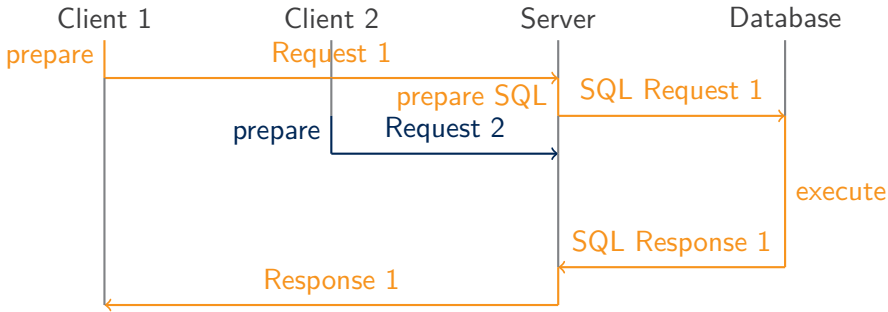

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

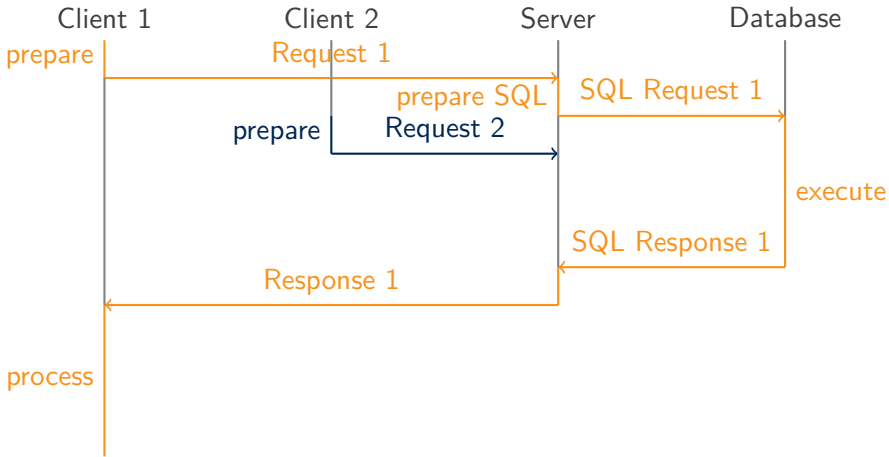

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

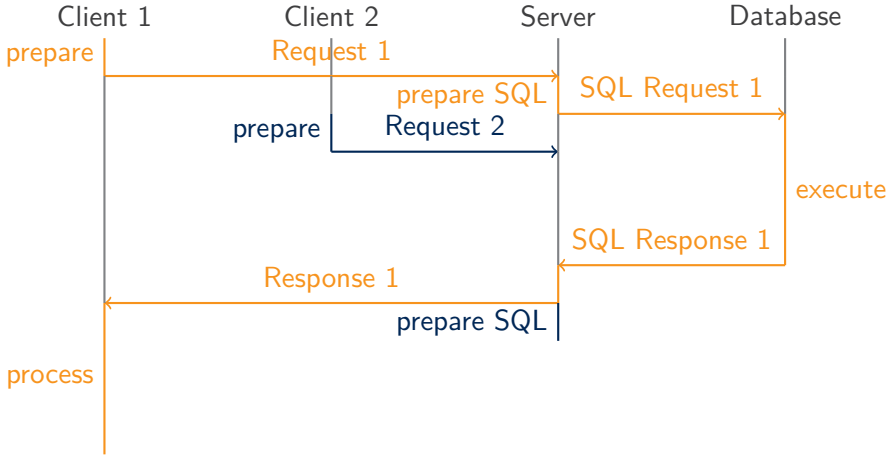

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

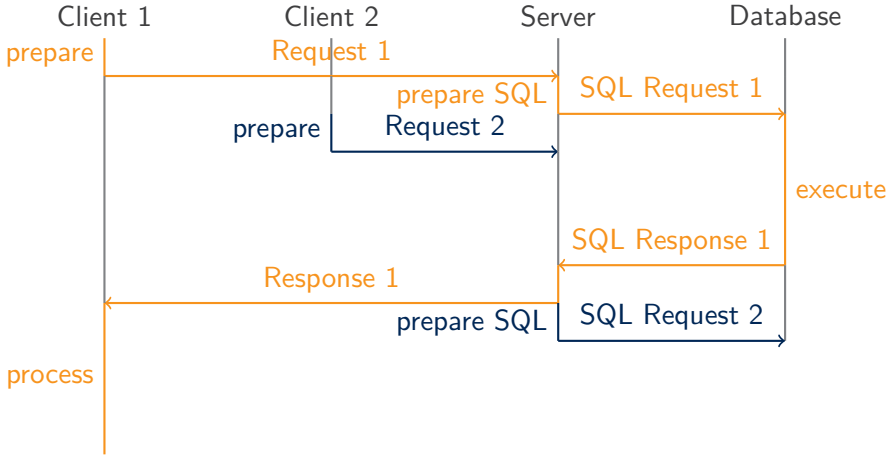

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

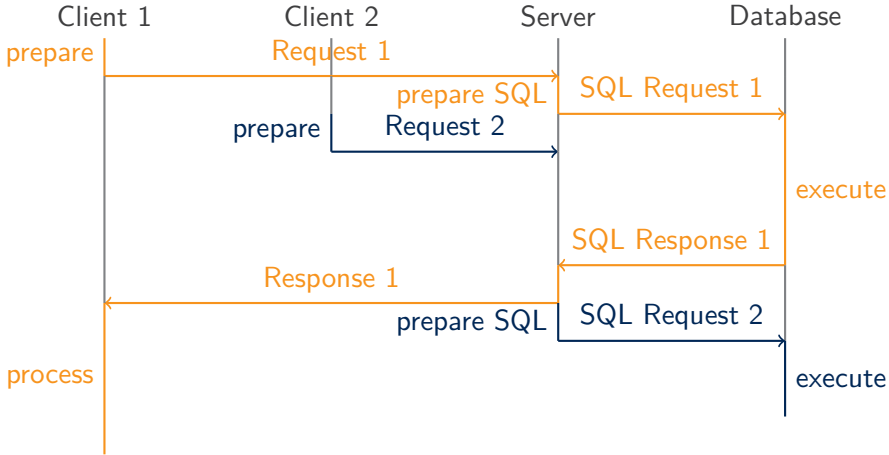

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

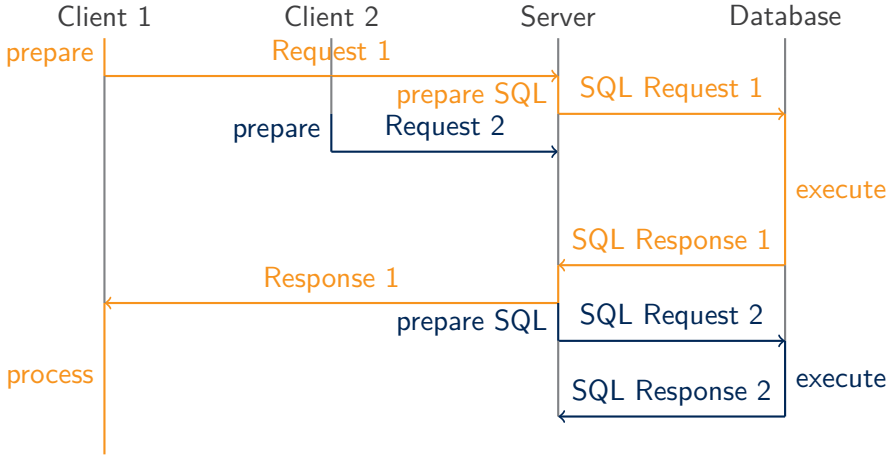

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

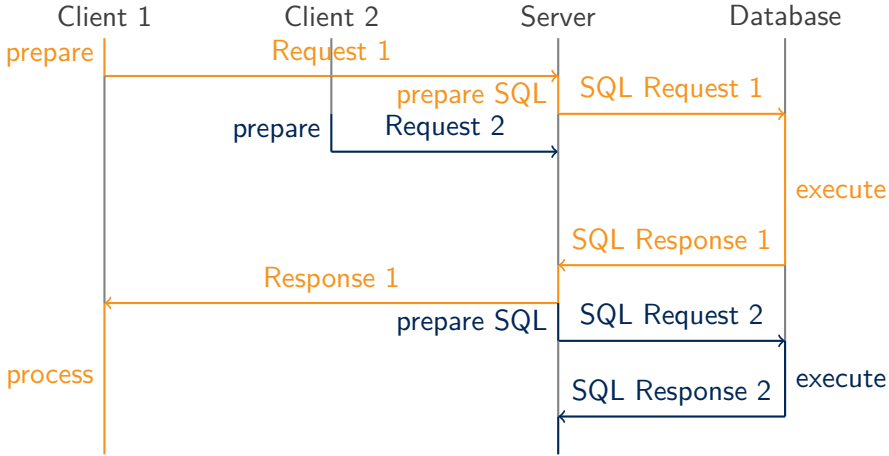

Abbildung: Sequenzdiagramm eines synchronen Servers

Synchroner Server

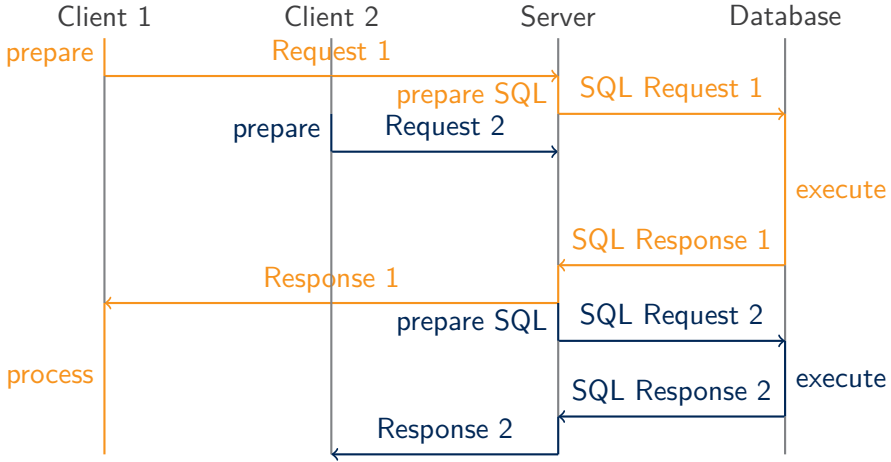

Abbildung: Sequenzdiagramm eines synchronen Servers

Asynchroner Server

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

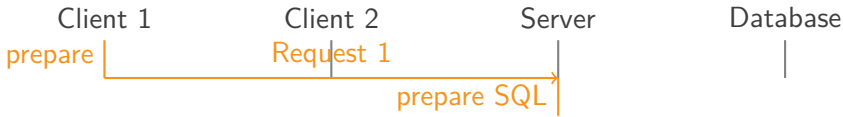

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

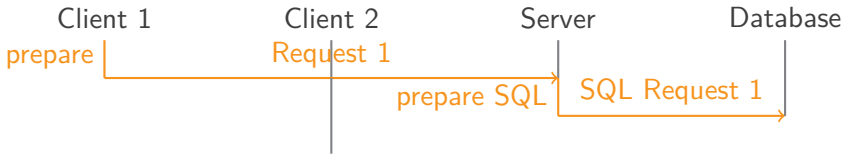

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

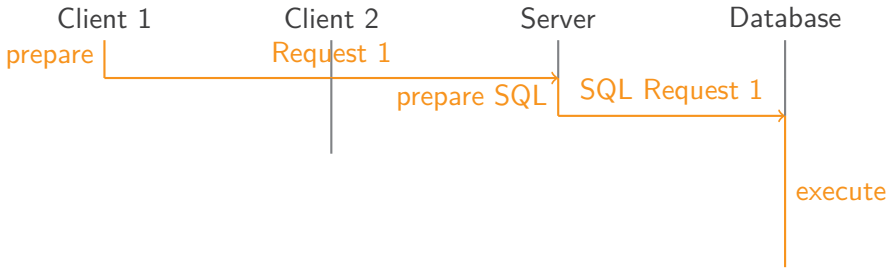

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

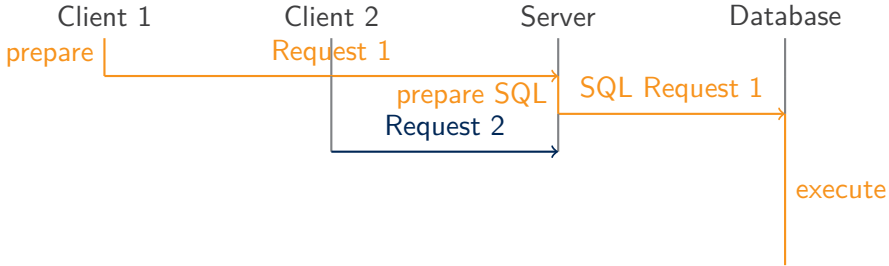

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

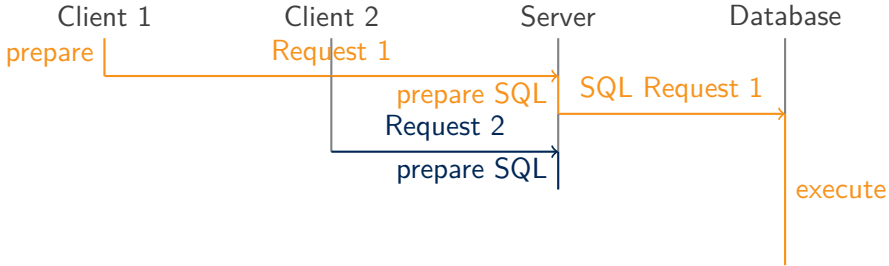

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

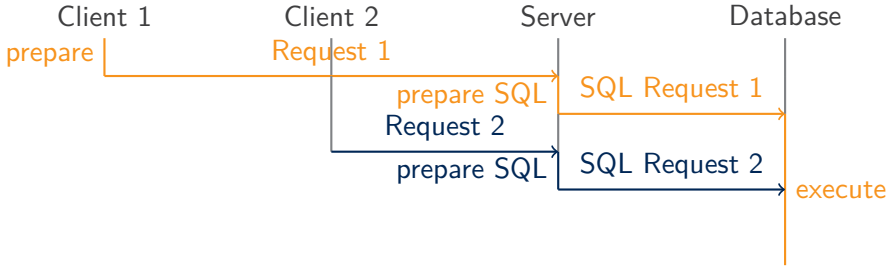

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

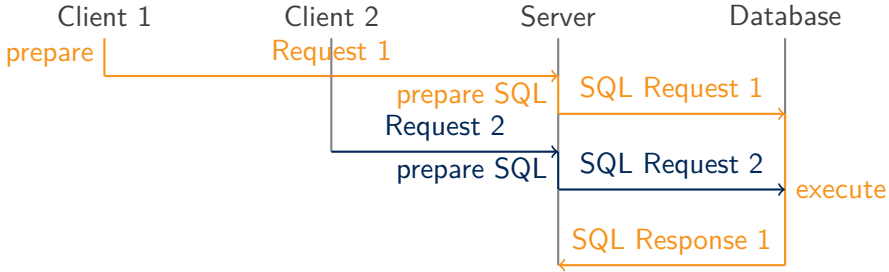

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

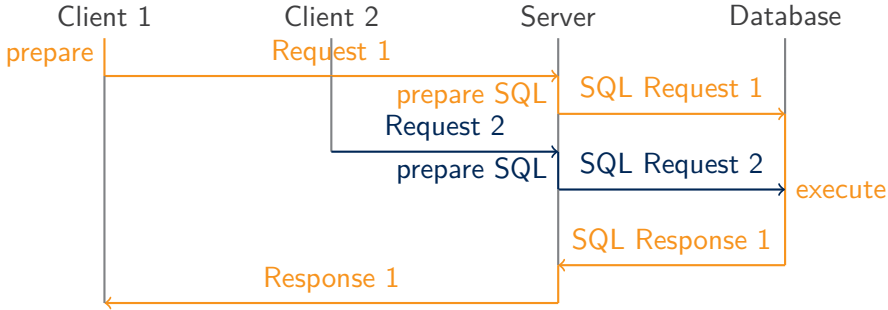

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

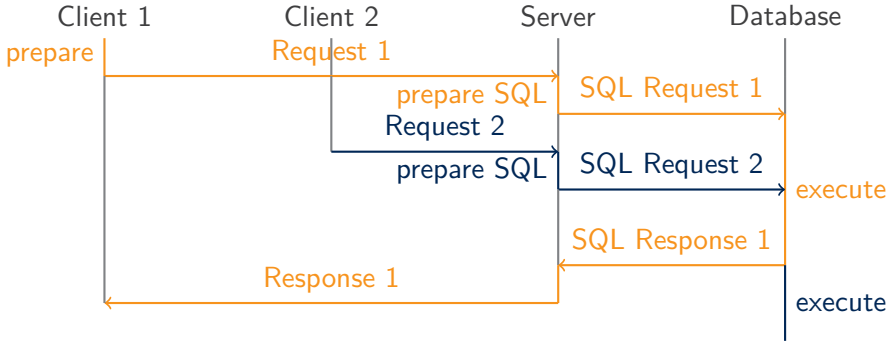

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

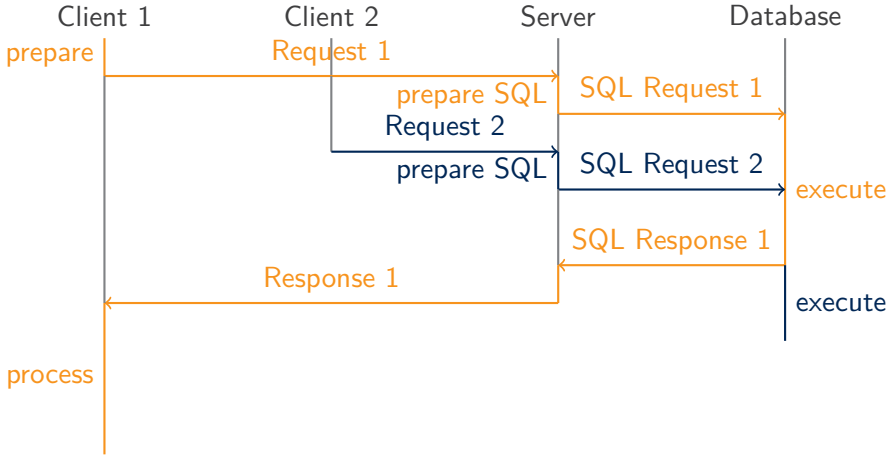

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

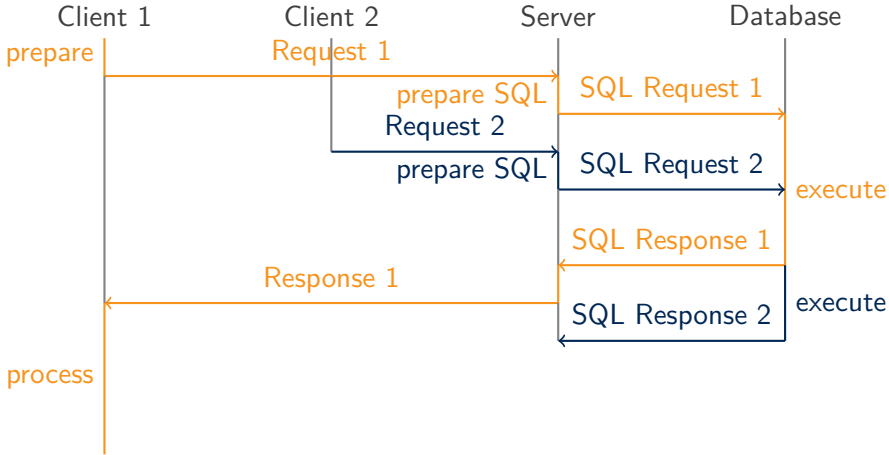

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

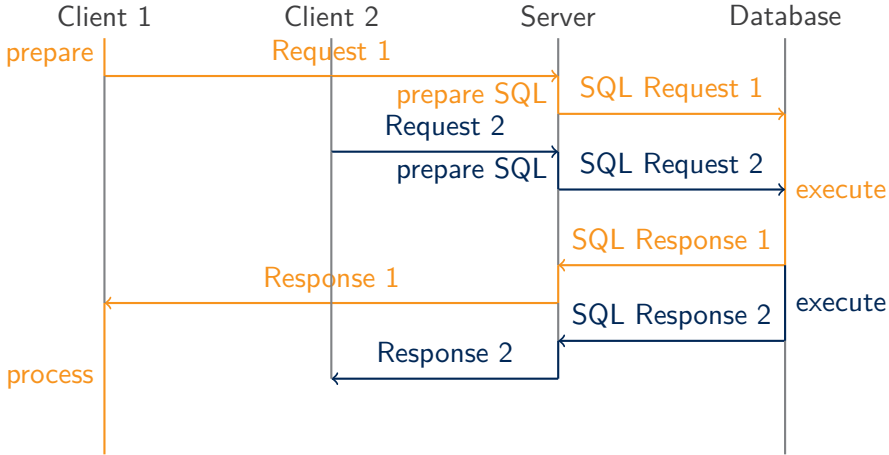

Abbildung: Sequenzdiagramm eines asynchronen Servers

Asynchroner Server

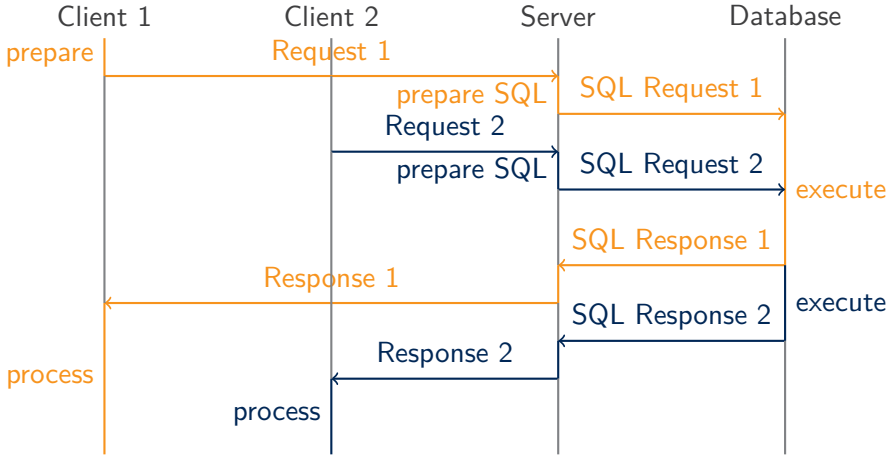

Abbildung: Sequenzdiagramm eines asynchronen Servers

Constrained Application Protocol (CoAP)

- Definiert im RFC 7252.

Constrained Application Protocol (CoAP)

- Definiert im RFC 7252.
- Entwickelt für **eingeschränkte** Umgebungen.

Constrained Application Protocol (CoAP)

- Definiert im RFC 7252.
- Entwickelt für **eingeschränkte** Umgebungen.
- Basiert auf einem **Anfragen-Antwort**-Modell zur Kommunikation.

Constrained Application Protocol (CoAP)

- Definiert im RFC 7252.
- Entwickelt für **eingeschränkte** Umgebungen.
- Basiert auf einem **Anfragen-Antwort**-Modell zur Kommunikation.
- Verwendung von **U**ser **D**atagram **P**rotocol (UDP).

Constrained Application Protocol (CoAP)

- Definiert im RFC 7252.
- Entwickelt für **eingeschränkte** Umgebungen.
- Basiert auf einem **Anfragen-Antwort**-Modell zur Kommunikation.
- Verwendung von **U**ser **D**atagram **P**rotocol (UDP).
- Alternativen: MQTT, CoAP over TCP, CoAP over Websockets.

Constrained Application Protocol (CoAP)

- Definiert im RFC 7252.
- Entwickelt für **eingeschränkte** Umgebungen.
- Basiert auf einem **Anfragen-Antwort**-Modell zur Kommunikation.
- Verwendung von **U**ser **D**atagram **P**rotocol (UDP).
- Alternativen: MQTT, CoAP over TCP, CoAP over Websockets.
- Implementierungen für verschiedene Programmiersprachen.

CoAP.NET

- Implementierung von CoAP für C# nach RFC 7252.

CoAP.NET

- Implementierung von CoAP für C# nach RFC 7252.
- Entwicklung inaktiv.

CoAP.NET

- Implementierung von CoAP für C# nach RFC 7252.
- Entwicklung inaktiv.
- Nur teilweise asynchron.

CoAP.NET

- Implementierung von CoAP für C# nach RFC 7252.
- Entwicklung inaktiv.
- Nur teilweise asynchron.
- Bietet eine Implementation für Client und Server.

Bachelorarbeit

- *»Hat die asynchrone Implementation eines Servers einen Einfluss auf dessen Durchsatzrate?«*

Bachelorarbeit

- *»Hat die asynchrone Implementation eines Servers einen Einfluss auf dessen Durchsatzrate?«* - **Nein**.

Bachelorarbeit

- *»Hat die asynchrone Implementation eines Servers einen Einfluss auf dessen Durchsatzrate?«* - **Nein**.
- Vollständige Neuimplementierung von CoAP.NET

Bachelorarbeit

- *»Hat die asynchrone Implementation eines Servers einen Einfluss auf dessen Durchsatzrate?«* - **Nein**.
- Vollständige Neuimplementierung von CoAP.NET **ausgenommen retransmission und block-wise transfer**.

Bachelorarbeit

- *»Hat die asynchrone Implementation eines Servers einen Einfluss auf dessen Durchsatzrate?«* - **Nein**.
- Vollständige Neuimplementierung von CoAP.NET **ausgenommen retransmission und block-wise transfer**.
- Implementierungen von Tests zur Messung des Durchsatzes.

Bachelorarbeit

- *»Hat die asynchrone Implementation eines Servers einen Einfluss auf dessen Durchsatzrate?«* - **Nein**.
- Vollständige Neuimplementierung von CoAP.NET **ausgenommen retransmission und block-wise transfer**.
- Implementierungen von Tests zur Messung des Durchsatzes.
- Vergleich zwischen synchroner und asynchroner Version.

Bachelorarbeit

- *»Hat die asynchrone Implementation eines Servers einen Einfluss auf dessen Durchsatzrate?«* - **Nein**.
- Vollständige Neuimplementierung von CoAP.NET **ausgenommen retransmission und block-wise transfer**.
- Implementierungen von Tests zur Messung des Durchsatzes.
- Vergleich zwischen synchroner und asynchroner Version.
- Quellcode frei verfügbar auf GitHub.

Bachelorarbeit

- *»Hat die asynchrone Implementation eines Servers einen Einfluss auf dessen Durchsatzrate?«* - **Nein**.
- Vollständige Neuimplementierung von CoAP.NET **ausgenommen retransmission und block-wise transfer**.
- Implementierungen von Tests zur Messung des Durchsatzes.
- Vergleich zwischen synchroner und asynchroner Version.
- Quellcode frei verfügbar auf GitHub.
- In Zusammenarbeit mit World-Direct eBusiness solutions GmbH.

Bachelorarbeit

- *»Hat die asynchrone Implementation eines Servers einen Einfluss auf dessen Durchsatzrate?«* - **Nein**.
- Vollständige Neuimplementierung von CoAP.NET **ausgenommen retransmission und block-wise transfer**.
- Implementierungen von Tests zur Messung des Durchsatzes.
- Vergleich zwischen synchroner und asynchroner Version.
- Quellcode frei verfügbar auf GitHub.
- In Zusammenarbeit mit World-Direct eBusiness solutions GmbH.
- Fokussierung auf Serialisierung und Deserialisierung als wichtigsten Baustein.

Bachelorarbeit

- *»Hat die asynchrone Implementation eines Servers einen Einfluss auf dessen Durchsatzrate?«* - **Nein**.
- Vollständige Neuimplementierung von CoAP.NET **ausgenommen retransmission und block-wise transfer**.
- Implementierungen von Tests zur Messung des Durchsatzes.
- Vergleich zwischen synchroner und asynchroner Version.
- Quellcode frei verfügbar auf GitHub.
- In Zusammenarbeit mit World-Direct eBusiness solutions GmbH.
- Fokussierung auf Serialisierung und Deserialisierung als wichtigsten Baustein.
- Implementation von CoAP over TCP zwecks Limitierung von UDP-Packets.

Implementierung

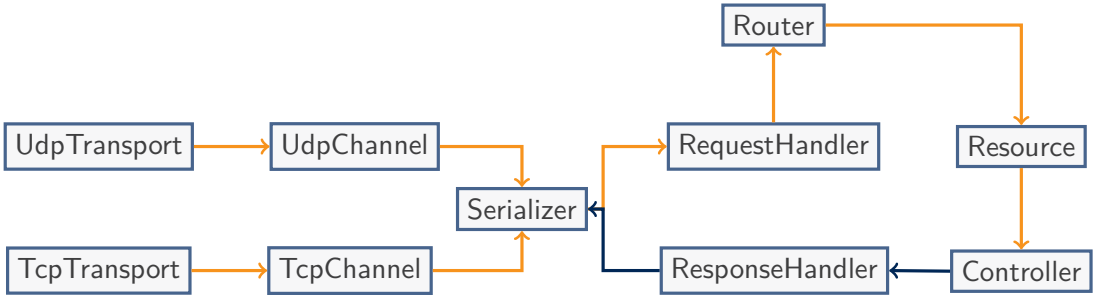

Abbildung: Überblick Softwarearchitektur

Implementierung

- `void CoapMessage Deserialize(ReadonlySpan<byte> value);`

Implementierung

- `void CoapMessage Deserialize(ReadonlySpan<byte> value);`
- `Task<CoapMessage> DeserializeAsync(Stream value, CancellationToken ct);`

Implementierung

- `void CoapMessage Deserialize(ReadonlySpan<byte> value);`
- `Task<CoapMessage> DeserializeAsync(Stream value, CancellationToken ct);`
- `void byte[] Serialize(CoapMessage message);`

Implementierung

- `void CoapMessage Deserialize(ReadonlySpan<byte> value);`
- `Task<CoapMessage> DeserializeAsync(Stream value, CancellationToken ct);`
- `void byte[] Serialize(CoapMessage message);`
- `Task SerializeAsync(Stream stream, CoapMessage message, CancellationToken ct);`

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.
- Nachrichtengröße durch Optionen und Payload definiert.

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.
- Nachrichtengröße durch Optionen und Payload definiert.
- Nachrichten zufällig generiert.

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.
- Nachrichtengröße durch Optionen und Payload definiert.
- Nachrichten zufällig generiert.
- Messung mehrere hundert Male wiederholt.

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.
- Nachrichtengröße durch Optionen und Payload definiert.
- Nachrichten zufällig generiert.
- Messung mehrere hundert Male wiederholt.
- Berechnung Mittelwert aus allen Durchläufen.

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.
- Nachrichtengröße durch Optionen und Payload definiert.
- Nachrichten zufällig generiert.
- Messung mehrere hundert Male wiederholt.
- Berechnung Mittelwert aus allen Durchläufen.
- Kennzahlen

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.
- Nachrichtengröße durch Optionen und Payload definiert.
- Nachrichten zufällig generiert.
- Messung mehrere hundert Male wiederholt.
- Berechnung Mittelwert aus allen Durchläufen.
- Kennzahlen
  - durchschnittliche Laufzeit

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.
- Nachrichtengröße durch Optionen und Payload definiert.
- Nachrichten zufällig generiert.
- Messung mehrere hundert Male wiederholt.
- Berechnung Mittelwert aus allen Durchläufen.
- Kennzahlen
  - durchschnittliche Laufzeit
  - Fehlertoleranz

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.
- Nachrichtengröße durch Optionen und Payload definiert.
- Nachrichten zufällig generiert.
- Messung mehrere hundert Male wiederholt.
- Berechnung Mittelwert aus allen Durchläufen.
- Kennzahlen
  - durchschnittliche Laufzeit
  - Fehlertoleranz
  - Standardabweichung

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.
- Nachrichtengröße durch Optionen und Payload definiert.
- Nachrichten zufällig generiert.
- Messung mehrere hundert Male wiederholt.
- Berechnung Mittelwert aus allen Durchläufen.
- Kennzahlen
  - durchschnittliche Laufzeit
  - Fehlertoleranz
  - Standardabweichung
  - allokierter Speicher

Serialisierung und Deserialisierung

- Laufzeitmessung von `Serialize`, `SerializeAsync`, `Deserialize` und `DeserializeAsync` mittels *BenchmarkDotnet*.
- Nachrichtengröße durch Optionen und Payload definiert.
- Nachrichten zufällig generiert.
- Messung mehrere hundert Male wiederholt.
- Berechnung Mittelwert aus allen Durchläufen.
- Kennzahlen
  - durchschnittliche Laufzeit
  - Fehlertoleranz
  - Standardabweichung
  - allokierter Speicher
  - Anzahl der Objekte in Gen 0 und Gen 1 Speichersegment

Ergebnis

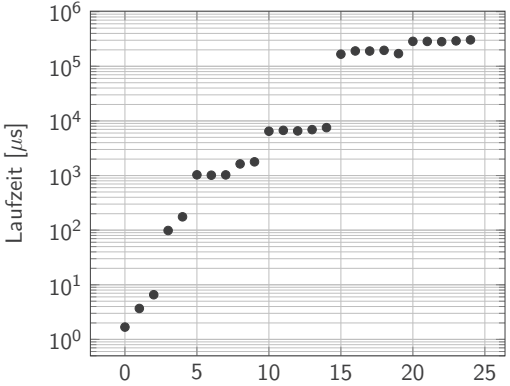

Abbildung: Messergebnisse von Deserialize-Methode als Diagramm

Ergebnis

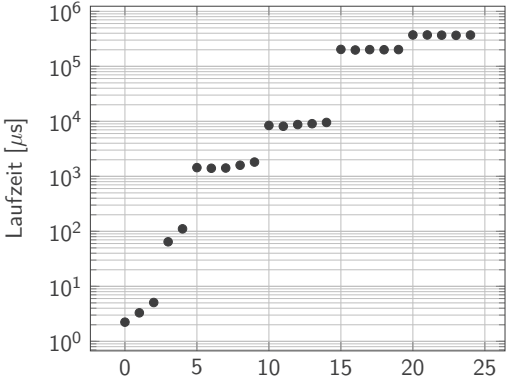

Abbildung: Messergebnisse von DeserializeAsync-Methode als Diagramm

Ergebnis

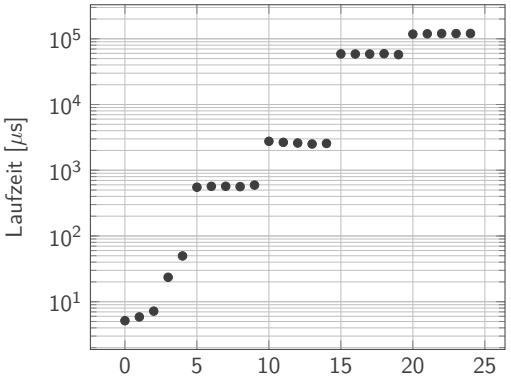

Abbildung: Messergebnisse von Serialize-Methode als Diagramm

Ergebnis

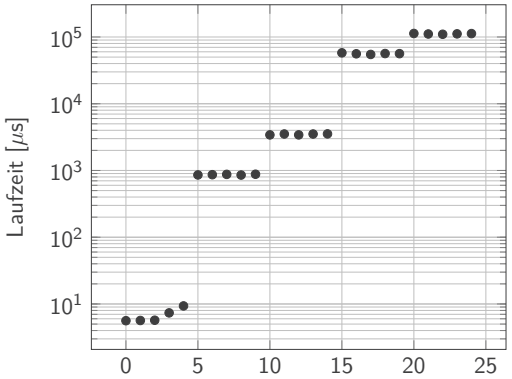

Abbildung: Messergebnisse von SerializeAsync-Methode als Diagramm

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.
- Geschwindigkeit von 4 B/s.

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.
- Geschwindigkeit von 4 B/s.
- Nachrichtengröße durch Anzahl der Optionen und Größe der Payload definiert.

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.
- Geschwindigkeit von 4 B/s.
- Nachrichtengröße durch Anzahl der Optionen und Größe der Payload definiert.
- Client und Server auf selbem System.

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.
- Geschwindigkeit von 4 B/s.
- Nachrichtengröße durch Anzahl der Optionen und Größe der Payload definiert.
- Client und Server auf selbem System.
- Kommunikation über *localhost* (127.0.0.1) mittels TCP.

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.
- Geschwindigkeit von 4 B/s.
- Nachrichtengröße durch Anzahl der Optionen und Größe der Payload definiert.
- Client und Server auf selbem System.
- Kommunikation über *localhost* (127.0.0.1) mittels TCP.
- Server öffnet zwei Ports

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.
- Geschwindigkeit von 4 B/s.
- Nachrichtengröße durch Anzahl der Optionen und Größe der Payload definiert.
- Client und Server auf selbem System.
- Kommunikation über *localhost* (127.0.0.1) mittels TCP.
- Server öffnet zwei Ports
  - 5683: Asynchrone Verarbeitung.

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.
- Geschwindigkeit von 4 B/s.
- Nachrichtengröße durch Anzahl der Optionen und Größe der Payload definiert.
- Client und Server auf selbem System.
- Kommunikation über *localhost* (127.0.0.1) mittels TCP.
- Server öffnet zwei Ports
  - 5683: Asynchrone Verarbeitung.
  - 5684: Synchrone Verarbeitung.

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.
- Geschwindigkeit von 4 B/s.
- Nachrichtengröße durch Anzahl der Optionen und Größe der Payload definiert.
- Client und Server auf selbem System.
- Kommunikation über *localhost* (127.0.0.1) mittels TCP.
- Server öffnet zwei Ports
  - 5683: Asynchrone Verarbeitung.
  - 5684: Synchrone Verarbeitung.
- Nachricht von Client an Server.

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.
- Geschwindigkeit von 4 B/s.
- Nachrichtengröße durch Anzahl der Optionen und Größe der Payload definiert.
- Client und Server auf selbem System.
- Kommunikation über *localhost* (127.0.0.1) mittels TCP.
- Server öffnet zwei Ports
  - 5683: Asynchrone Verarbeitung.
  - 5684: Synchrone Verarbeitung.
- Nachricht von Client an Server.
- Fünf voneinander unabhängige Durchläufe.

Nachrichtenübetragung

- Simulation einer langsamen Client-Server-Kommunikation.
- Geschwindigkeit von 4 B/s.
- Nachrichtengröße durch Anzahl der Optionen und Größe der Payload definiert.
- Client und Server auf selbem System.
- Kommunikation über *localhost* (127.0.0.1) mittels TCP.
- Server öffnet zwei Ports
  - 5683: Asynchrone Verarbeitung.
  - 5684: Synchrone Verarbeitung.
- Nachricht von Client an Server.
- Fünf voneinander unabhängige Durchläufe.
- Laufzeit: Start bis Ende der Übertragung.

Ergebnis

- Asynchrone Verarbeitung skaliert mit steigender Nachrichtengröße.

Ergebnis

- Asynchrone Verarbeitung skaliert mit steigender Nachrichtengröße.
- Differenz zwischen Synchron und Asynchron nur einige hundert Millisekunden bis Sekunden.

Fazit

- Einsatz von Asynchronität abhängig von Problemstellung - **allgemein jedoch die bessere Wahl**.

Fazit

- Einsatz von Asynchronität abhängig von Problemstellung - **allgemein jedoch die bessere Wahl**.
- Verarbeitungsgeschwindigkeit von Nachrichtengröße abhängig.

Fazit

- Einsatz von Asynchronität abhängig von Problemstellung - **allgemein jedoch die bessere Wahl**.
- Verarbeitungsgeschwindigkeit von Nachrichtengröße abhängig.
- CoAP scheint nicht von Asynchronität zu profitieren.

Ausblick

- Implementierung von **retransmission** und **block-wise transfer**.

Ausblick

- Implementierung von **retransmission** und **block-wise transfer**.
- Implementierung **CoAP over TCP, Websockets und TLS**.

Ausblick

- Implementierung von **retransmission** und **block-wise transfer**.
- Implementierung **CoAP over TCP, Websockets und TLS**.
- Verbesserung der Nachrichtenverarbeitung.

Ausblick

- Implementierung von **retransmission** und **block-wise transfer**.
- Implementierung **CoAP over TCP, Websockets und TLS**.
- Verbesserung der Nachrichtenverarbeitung.
- Implementation von einigen *syntactic sugar* Funktionalitäten aus ASP.NET Core.