HAW HAMBURG

Bachelorarbeit

Sam Taboubi

Analyse der Service Discovery von DDS und SOME/IP in Automotive-Netzwerken

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

Sam Taboubi

Analyse der Service Discovery von DDS und SOME/IP in Automotive-Netzwerken

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Jan Sudeikat

Eingereicht am: 14. November 2025

Sam Taboubi

Thema der Arbeit

Analyse der Service Discovery von DDS und SOME/IP in Automotive-Netzwerken

Stichworte

Fahrzeugnetzwerke, Middleware, Dienstorientierte Architektur, Mininet, SOME/IP, DDS, Service Discovery

Kurzzusammenfassung

DDS und SOME/IP finden als Middleware-Protokolle unterschiedlichen Ursprungs immer weiter Verbreitung im Automotive-Bereich. Während SOME/IP speziell für den Einsatz in Fahrzeugnetzwerken entwickelt wurde, stammt DDS ursprünglich aus dem IIOT-Umfeld. Beide Protokolle bieten Mechanismen zur dynamischen Service Discovery, die es ermöglichen, Dienste in einem Fahrzeugnetzwerk zu finden und zu nutzen. In dieser Arbeit wird eine automatisierte Testumgebung auf Basis von Mininet entwickelt, um die Service Discovery-Leistung der Implementierungen CycloneDDS und vSomeIP anhand typischer Anforderungen aus dem Automotive-Bereich zu vergleichen und zu bewerten. Hierzu werden verschiedene Testszenarien entworfen, in denen Teilnehmerkonstellationen und Konfigurationsparameter variiert werden. Die Analyse zeigt, dass vSomeIP durchgängig ressourceneffizienter arbeitet und niedrigere Discovery-Latenzen erreicht als CycloneDDS. Zudem lassen sich durch die Konfigurationsparameter in vSomeIP gezielte Optimierungen bzgl. des Discovery-Verhaltens erzielen. Außerdem zeigt sich, dass die Testumgebung bei CycloneDDS in anspruchsvollen Szenarien an ihre Leistungsgrenzen stößt, was die Bewertung der Skalierbarkeit erschwert. Abschließend zeigt die Arbeit den weiteren Bedarf nach realitätsnahen Tests auf und liefert ein Grundgerüst für die Evaluation des Netzwerkverkehrs in anderen Testumgebungen.

Sam Taboubi

Title of Thesis

Analyzing DDS and SOME/IP service discovery in automotive networks

Keywords

In-vehicle networks, middleware, service-oriented architecture, Mininet, SOME/IP, DDS, Service Discovery

Abstract

DDS and SOME/IP, both middleware protocols with different origins, are becoming increasingly widespread in the automotive sector. While SOME/IP was developed specifically for use in vehicle networks, DDS originates from the IIOT environment. Both protocols offer mechanisms for dynamic service discovery, which enable services to be found and used in a vehicle network. In this work, an automated test environment based on Mininet is developed to compare and evaluate the service discovery performance of the CycloneDDS and vSomeIP implementations using typical requirements from the automotive sector. For this purpose, various test scenarios are designed in which participant constellations and configuration parameters are varied. The analysis shows that vSomeIP consistently operates more resource-efficiently and achieves lower discovery latencies than CycloneDDS. In addition, the configuration parameters in vSomeIP allow for targeted optimizations with regard to discovery behavior. It shows that the CycloneDDS test environment reaches its performance limits in demanding scenarios, which makes it difficult to evaluate scalability. Finally, the thesis highlights the need for more realistic tests and provides a basic framework for evaluating network traffic in other test environments.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADAS Advanced Driver Assistance System

AUTOSAR Automotive Open System Architecture

COVESA Connected Vehicle Systems Alliance

DDS Data Distribution Service

DNS Domain Name System

ECU Electronic Control Unit

GUID Globally Unique Identifier

IIOT Industrial Internet of Things

IVN In-Vehicle Network

JSON Javascript Object Notation

MKQ Methode der kleinsten Quadrate

OMG Object Management Group

QoS Quality of Service

ROS2 Robot Operating System 2

RPC Remote Procedure Call

RTPS Real-Time Publish Subscribe Protocol

SD Service Discovery

SDV Software Defined Vehicle

SEDP Simple Endpoint Discovery Protocol

SOA Service Oriented Architecture

SoC System-on-a-Chip

SOME/IP Scalable Service-Oriented Middleware over IP

SOME/IP-SD SOME/IP Service Discovery

SPDP Simple Participant Discovery Protocol

TCP Transmission Control Protocol

TSN Time Sensitive Networking

UDP User Datagram Protocol

1 Einleitung

Die elektrisch/elektronische Architektur (E/E-Architektur) moderner Fahrzeuge durchläuft derzeit einen tiefgreifenden Wandel hin zu flexibleren, softwaredefinierten Systemen, den sogenannten *Software Defined Vehicle*s (SDVs). Durch Entwicklungen wie *Advanced Driver Assistance System*s (ADAS) und komplexe Infotainment-Systeme weisen moderne Fahrzeuge eine immer größer werdende Anzahl an Software-Komponenten auf, welche auf verschiedenen *Electronic Control Unit*s (ECUs) laufen. Klassische, statisch konfigurierte Netzwerktopologien sind dadurch nur noch äußerst schwer zu handhaben, weshalb statische, Feldbus-basierte Kommunikationsmuster immer häufiger durch dynamische, Ethernet-basierte Methoden abgelöst werden. Dies resultiert im Konzept des SDV, welches durch eine lose Kopplung und dynamische Kommunikation zwischen den Komponenten gekennzeichnet ist [6, 14, 31]. Ein zentraler Aspekt von SDVs ist die *Service Oriented Architecture* (SOA), bei der Funktionalitäten als wiederverwendbare Dienste bereitgestellt werden. Diese Dienste kommunizieren über definierte Schnittstellen, was die Modularität und Skalierbarkeit des Systems erhöht. Ethernet-basierte Netzwerke spielen hier eine entscheidende Rolle, da sie die notwendige Bandbreite und Flexibilität für ein hohes Maß an Netzwerkverkehr bieten. Sichergestellt wird diese Kommunikation durch Middlewares, welche die einzelnen Services in einer SOA voneinander entkoppeln und eine dynamische *Service Discovery* (SD) bieten, was eine High-level-Abstraktion der verschiedenen realen Endpunkte ermöglicht.

Im Automobilbereich ist hier als Middleware insbesondere *Scalable Service-Oriented Middleware over IP* (SOME/IP) zu nennen, welche im Kontext von *Automotive Open System Architecture* (AUTOSAR) entwickelt wurde und eine standardisierte Kommunikation zwischen ECUs im Fahrzeug ermöglicht. SOME/IP unterstützt dabei sowohl SD als auch die Kommunikation über IP-basierte Netzwerke und bildet somit die zentrale Grundlage für serviceorientierte Architekturen in vielen modernen Fahrzeugen. SOME/IP setzt hierbei auf geringen Overhead und eine effiziente Nachrichtenübertragung, um den Anforderungen an Echtzeitfähigkeit und Ressourcenbeschränkungen in Fahrzeugen gerecht zu werden.

Data Distribution Service (DDS) stellt eine weitere Middleware dar, welche immer häufiger Verwendung im Automotive-Bereich findet. Ursprünglich aus Bereichen wie *Industrial Internet of Things* (IIOT) und Robotik stammend, basiert sie auf einem verteilten, datenzentrischen Ansatz. Im Gegensatz zu klassischen, serviceorientierten Architekturen fokussiert sich DDS auf die effiziente und flexible Verteilung von Daten zwischen Teilnehmern, ohne dass diese explizit voneinander wissen müssen. Über *Quality of Service* (QoS)-Parameter lässt sich die Kommunikation in DDS gezielt an die Anforderungen der jeweiligen Anwendung anpassen, beispielsweise hinsichtlich Latenz, Zuverlässigkeit oder Bandbreite.

An beide Middlewares werden hohe Anforderungen bezüglich der Service Discovery gestellt: Innerhalb des Fahrzeugs müssen Services dynamisch gefunden, verbunden und ggf. auch wieder getrennt werden können, ohne dass ihre Verbindungen statisch vorab definiert sind. Dies ist insbesondere vor dem Hintergrund der zunehmenden Komplexität und Modularität von Fahrzeugsystemen essenziell, da neue Funktionen und Steuergeräte flexibel integriert werden müssen. Außerdem erfordert es robuste Mechanismen zur Service Discovery und -verwaltung, die sowohl Skalierbarkeit als auch Zuverlässigkeit gewährleisten [15].

Diese Arbeit hat das Ziel, die Service Discovery-Mechanismen von SOME/IP und DDS systematisch zu analysieren und zu vergleichen. Im Mittelpunkt steht die quantitative Bewertung ihrer Effizienz, Performanz und Skalierbarkeit in einer simulierten Umgebung. Dazu werden folgende Teilziele verfolgt:

- Empirische Untersuchung der Service Discovery beider Middlewares unter definierten Szenarien aus dem Automotive-Kontext.
- Identifikation von Stärken und Schwächen der jeweiligen Protokolle in Bezug auf Latenz, Netzwerkauslastung und Ressourcenverbrauch.
- Vergleich und Bewertung der Protokolle anhand nicht-funktionaler Anforderungen wie Discovery-Zeit und Nachrichtenaufkommen.

Diese Arbeit beschränkt sich auf eine quantitative Untersuchung der Service Discovery von SOME/IP und DDS in einem Emulationskontext mit einer einfachen Topologie. Folgende Aspekte sind hierbei zu beachten:

- Da der Fokus auf nicht-funktionalen Anforderungen liegt, werden funktionale Anforderungen nur implizit bewertet.

- Sicherheitsmechanismen (z. B. Authentifizierung, Verschlüsselung) werden nicht betrachtet.
- Es wird die Service Discovery für eventbasierte Kommunikation (Publish/Subscribe) untersucht; *Remote Procedure Call* (RPC)-Mechanismen liegen nicht im Fokus.
- Die Nachrichtenübertragung nach erfolgreicher Discovery (z. B. Datenrate, QoS-Policies) wird nicht analysiert.
- Die Implementierungen CycloneDDS (für DDS) und vSomeIP (für SOME/IP) dienen als beispielhafte Untersuchungsgegenstände; andere Implementierungen bedürfen separater Untersuchungen.

Die Arbeit ist wie folgt aufgebaut: Kapitel 2 vermittelt die theoretischen Grundlagen zu SOME/IP (inkl. vSomeIP) und DDS (inkl. RTPS und CycloneDDS) sowie die Vergleichskriterien für die Service Discovery. Kapitel 3 beleuchtet den Stand der Forschung durch verwandte Arbeiten zu Middlewares in Fahrzeugen. Kapitel 4 gibt einen Überblick über die nicht-funktionellen Anforderungen an die Service Discovery. Kapitel 5 beschreibt den Versuchsaufbau, die Messmethoden, getestete Konfigurationsparameter und definiert Testszenarien. Kapitel 6 präsentiert die Ergebnisse der Evaluation und diskutiert diese. Kapitel 7 fasst die Arbeit zusammen, zeigt weiteren Forschungsbedarf auf und gibt einen Ausblick.

2 Begriffsbildung und Grundlagen

Im folgenden Kapitel werden die in der Arbeit verwendeten Begriffe und Diagrammnotationen erläutert sowie anschließend die grundlegende Funktionsweise der Service Discovery jeweils in DDS und SOME/IP erläutert.

Begriffsdefinitionen DDS und SOME/IP verwenden aufgrund ihrer unterschiedlichen Ursprünge und Anwendungsbereiche teils unterschiedliche Terminologien für ähnliche Konzepte. Während DDS den Begriff *Publisher* für eine Entität verwendet, die Daten veröffentlicht, und *Subscriber* für eine Entität, die Daten abonniert, nutzt SOME/IP die Begriffe *Service* für den Anbieter von Diensten und *Client* für den Konsumenten von Diensten. In technologieübergreifenden Kontexten werden in dieser Arbeit zur Vereinheitlichung der Terminologie und Erhöhung der Lesbarkeit folgende eindeutige Bezeichnungen verwendet:

Subscriber Bezeichnet sowohl einen *Subscriber* (im DDS-Kontext) als auch einen *Client* (im SOME/IP-Kontext). Ein Subscriber ist eine Instanz, die Daten abonniert bzw. Dienste konsumiert.

Publisher Bezeichnet sowohl einen *Publisher* (im DDS-Kontext) als auch einen *Service* (im SOME/IP-Kontext). Ein Publisher ist eine Instanz, die Daten publiziert bzw. Dienste anbietet.

Notation in Sequenzdiagrammen In dieser Arbeit werden zur Veranschaulichung der Abläufe in der Service Discovery UML-Sequenzdiagramme verwendet. Die Diagramme folgen dabei den folgenden Konventionen:

- Da der Fokus in dieser Arbeit auf dem Ermitteln von Zeitstempeln beim Senden und Empfangen von Nachrichten liegt, wird zugunsten der Übersichtlichkeit auf die Darstellung von Aktivierungsbalken verzichtet.

- UML-Sequenzdiagramme bieten keine Möglichkeit, Multicast-Nachrichten nativ abzubilden. In den Diagrammen dieser Arbeit ist daher eine separate Lebenslinie für Multicast-Kommunikation enthalten, an die (1) Multicast-Nachrichten gesendet und von der aus (2) Multicast-Nachrichten an die Empfänger weitergeleitet werden.
- Da der Nachrichtenaustausch in den Sequenzdiagrammen dieser Arbeit i.d.R. asynchron ist, stellen die Abläufe meist nur beispielhaft eine mögliche Reihenfolge der Nachrichten dar.
- Zeitstempel in Sequenzdiagrammen werden als farbige Annotationen an Nachrichtenpfeilen dargestellt. Die für den Teilnehmer relevanten Zeitstempel sind farblich einheitlich markiert.
- Zeitintervalle werden als farbige Balken über den Lebenslinien sowie einer farbigen Annotation dargestellt.

2.1 DDS und SOME/IP im Automotive-Kontext

SDVs weisen eine immer größer werdende Anzahl von Steuergeräten auf, weshalb statisch konfigurierte Netzwerke im Fahrzeug die dadurch immer weiter steigende Datenmenge nicht mehr angemessen bewältigen können. Neue E/E-Architekturen sind geprägt von zonalen und domänenbasierten Architekturen, in den zentrale Knoten (SoCs) Rechenkapazitäten übernehmen [18]. Mit dieser Entwicklung geht auch die wachsende Bedeutung von Echtzeitaspekten einher: Sicherheitskritische Daten müssen innerhalb fester Deadlines zwischen ECUs ausgetauscht werden. Hierbei spielt (Automotive) Ethernet in Verbindung mit *Time Sensitive Networking* (TSN) aus dem IEEE 802.1Q-Standard [13] eine immer größere Rolle, da es perspektivisch Echtzeitfähigkeit in Ethernet-Netze bringt. DDS und SOME/IP sind zwei Middlewares, die in diesem Kontext eine zentrale Rolle spielen, da sie dynamische Service Discovery nutzen und somit die Kommunikation zwischen den verschiedenen Steuergeräten in einem Fahrzeug ermöglichen.

2.1.1 DDS

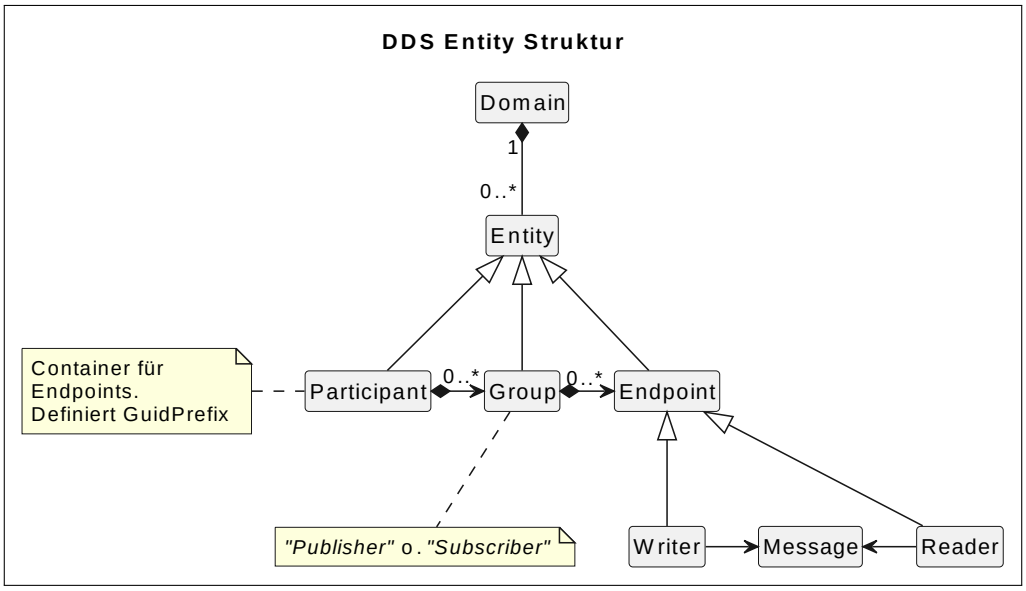
DDS[1] ist ein von der *Object Management Group* (OMG) entwickelter Standard für die Kommunikation in verteilten, hochdynamischen Systemen. Er findet Verwendung in Bereichen wie IIOT, Raumfahrt und Robotik sowie immer häufiger im Automotive-Kontext. DDS liegt ein datenzentrischer Ansatz zugrunde: die Kommunikationsebene ist für die Teilnehmer transparent, sodass ein „global data space“ [20], eine verteilte Datenbank, simuliert wird. DDS bietet umfangreiche Unterstützung für verschiedene QoS-Attribute und Security-Funktionen und ist mittlerweile in den *AUTOSAR Foundation*-Standard integriert [24].

DDS Architektur

Abbildung 2.1 zeigt die grundlegende Architektur der wichtigsten Entitäten in DDS, welche im folgenden Abschnitt referenziert werden. In DDS gehören alle Entitäten exakt einer *Domain* an. Die Domain bildet einen virtuellen Kommunikationskontext: Nur Entitäten innerhalb derselben Domain können miteinander kommunizieren. Sie kann für jede Anwendung statisch gesetzt oder dynamisch generiert werden. Alle in der Domain sichtbaren Entitäten erben von einer Basisklasse *Entity*, welche einen *Globally Unique Identifier* (GUID) besitzt. Darauf basierend existieren *Endpoints*, welche Quelle oder Ziel von Nachrichten sein können. Endpunkte sind einer *Group* zugeordnet: Die Group *Publisher* beinhaltet *Writer*-Endpunkte, während die Group *Subscriber Reader*-Endpunkte enthält. Writer sind für das Senden von *Messages* zuständig, während Reader Messages empfangen. Jeder Endpunkt ist genau einem *Topic* zugeordnet, welches den Namen und den Typ der ausgetauschten Daten definiert. Die Endpunkte können entweder die eigens definierten, fachlichen Datenendpunkte als auch spezielle Endpunkte für die Discovery sein. *Participants* sind Container für diese Endpunkte, welche das Prefix jeder GUID vorgeben. Dadurch kann jede Entity durch ihre GUID eindeutig einem Participant zugeordnet werden. Jede Anwendung definiert genau einen Participant, wodurch sie auch eindeutig in der Domain identifiziert werden kann.

[1] Die Grundlage für die Erklärungen bildet die Spezifikation für RTPS 2.5 [22] sowie das offizielle DDS Foundation Wiki [19]

Abb. 2.1: DDS Entity Architektur (adaptiert von [22, Abb. 7.2])



DDS Service Discovery

Real-Time Publish Subscribe Protocol (RTPS) ist ein von der OMG eigens spezifiziertes Interoperabilitätsprotokoll für DDS [22] und definiert, wie DDS-Implementierungen auf Netzwerkebene miteinander kommunizieren. Die Entitäten aus DDS werden exakt auf RTPS-Entitäten abgebildet. Das Protokoll erlaubt den Einsatz herstellerspezifischer Discovery-Protokolle, erfordert jedoch von jeder Implementation, mindestes die *Simple Discovery* zu unterstützen, um herstellerunabhängige Interoperabilität zu gewährleisten. Simple Discovery besteht aus zwei Bestandteilen: Dem *Simple Participant Discovery Protocol* (SPDP) und dem *Simple Endpoint Discovery Protocol* (SEDP), welche einen zweistufigen Ablauf in der Discovery-Phase vorgeben.

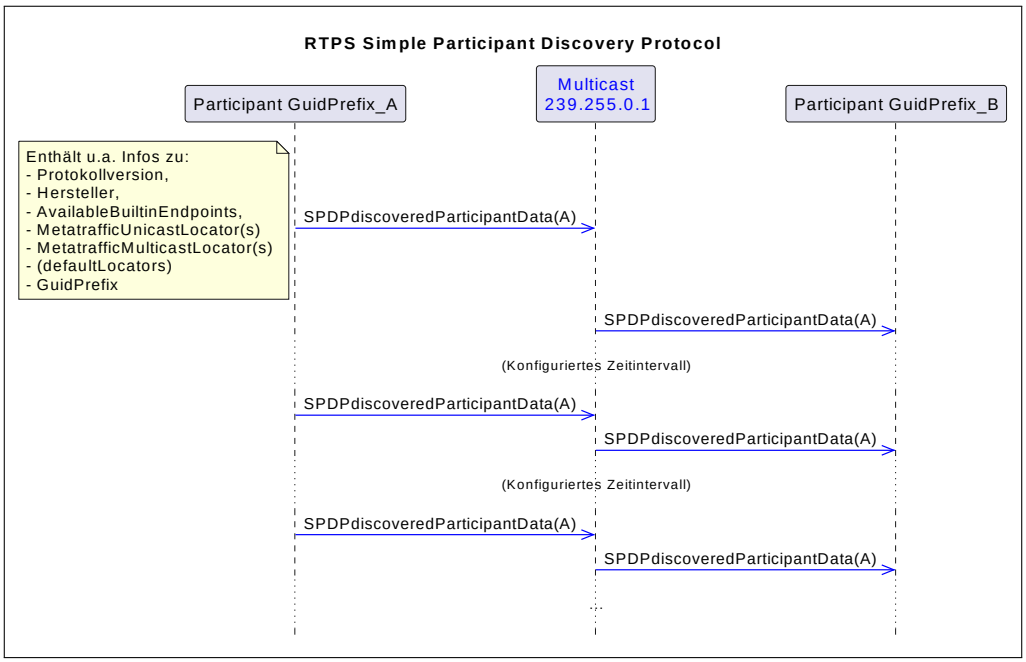
1. **SPDP** Participants geben via Multicast Metainformationen (z. B. Protokollversion, Netzwerkadressen) über einen *Well-known Port* bekannt.
2. **SEDP** Participants geben via Multicast oder Unicast ihre verfügbaren Endpunkte bekannt.

Durch SPDP werden Informationen darüber ausgetauscht, welche *Participants* in der RTPS-Domain verfügbar sind und welche *builtinEndpoints* sie zur Verfügung stellen. Dies sind spezielle vordefinierte Endpunkte, welche Meta-Informationen enthalten, etwa

ob ein Teilnehmer *Writer* und/oder *Reader* besitzt und wie diese zu erreichen sind. Basierend auf dieser Information kann ein Empfänger im nachfolgenden SEDP-Prozess ausschließlich relevante Daten zu seinen Endpunkten senden.

Enthält die SPDP-Nachricht eines Teilnehmers bspw. keinen SEDP-Endpunkt des Typs `PUBLICATIONS_DETECTOR`, so kann der Empfänger darauf verzichten, in der folgenden SEDP-Phase seine *Publications* zu übermitteln. Andersherum erzeugt jede Seite nur die Endpunkte, die sie braucht (z. B. reine Writer-Teilnehmer nur `PUBLICATIONS_ANNOUNCER` und `SUBSCRIPTIONS_DETECTOR`). Abb. 2.2 zeigt den Nachrichtenaustausch in der SPDP-Phase.

Abb. 2.2: RTPS Simple Participant Discovery



In der SEDP-Phase werden je nach Endpunkt-Verfügbarkeit Datensätze zu *Publications* und/oder *Subscriptions* ausgetauscht. Diese Datensätze enthalten insbesondere Informationen zu Unicast-/Multicast-Adressen (*Locators*), Topic-Namen und QoS-Attributen. Der Writer entscheidet in der Folge implementationsabhängig zur Laufzeit, welche zur Verfügung stehenden Locators er zum Erreichen des fachlichen Endpunkts verwendet.

Abb. 2.3 zeigt den Nachrichtenaustausch in der SEDP-Phase von einem reinen Publisher zu einem reinen Subscriber: die Nachricht enthält Endpunktinformationen vom Typ *DiscoveredWriterData*, welche die je Endpunkt angebotene Topic inkl. spezifischer QoS-

Attribute beinhaltet. Weitergehend sind hier auch Writer-GUID sowie die Locators des fachlichen Endpunkts enthalten, über die der Writer seine Daten schickt.

Abb. 2.3: RTPS Simple Endpoint Discovery: Publisher zu Subscriber

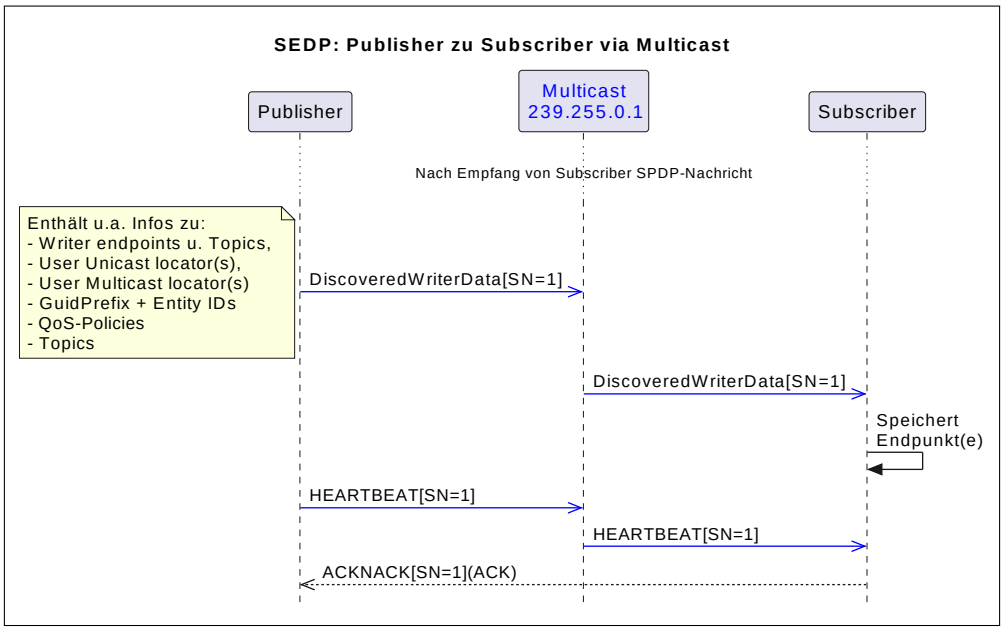
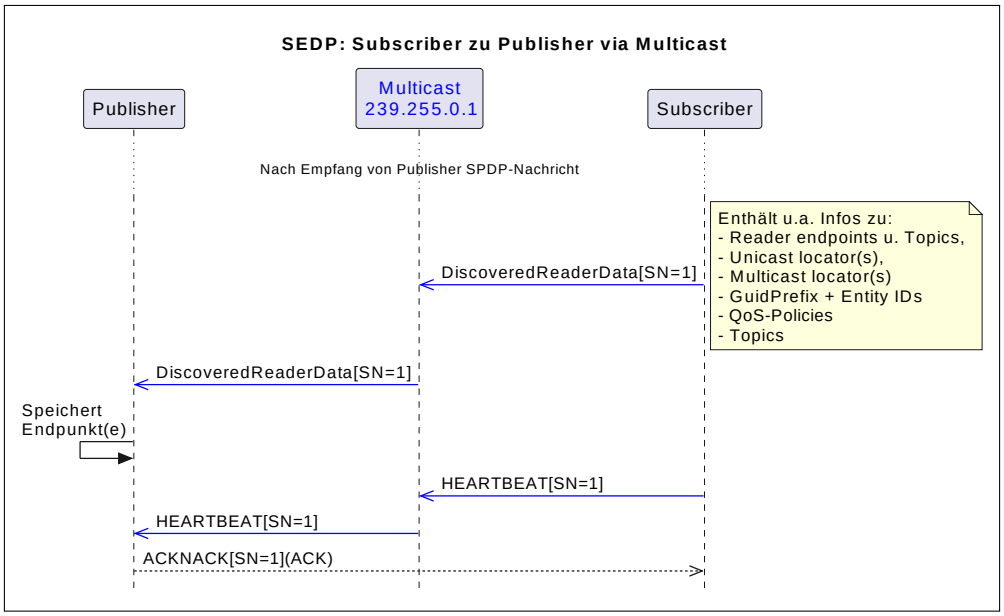


Abb. 2.4 zeigt den umgekehrten Fall von einem reinen Subscriber zu einem reinen Publisher: die Nachricht enthält Endpunktinformationen vom Typ *DiscoveredReaderData*, welche je Endpunkt die gesuchte Topic inkl. spezifischer QoS-Attribute beinhaltet. Weitergehend sind hier auch Reader-GUID sowie die Locators des fachlichen Endpunkts enthalten, über die der Reader seine Daten empfängt.

Die Kommunikation der SEDP-Nachrichten ist verlässlich, d. h. verlorene Nachrichten werden erneut übertragen. Der Mechanismus dahinter wird auf Applikationsebene durch `HEARTBEAT`- und `ACKNACK`-Nachrichten realisiert: `HEARTBEAT`s werden von einem Writer periodisch gesendet, um den Readern den Stand der bisher gesendeten Daten mitzuteilen. Da alle Nachrichten in RTPS eine fortlaufende Sequenznummer haben, können Nachrichten darüber identifiziert werden. Reader teilen über ein `ACKNACK` mit, ob ihnen Nachrichten fehlen und wenn ja, um welche Sequenznummern es sich handelt. Abb. 2.5 veranschaulicht dies: Der Writer für SPDP-Nachrichten sendet eine Nachricht, wodurch der Sequenzzähler auf 1 erhöht wird. Der Reader empfängt diese Nachricht jedoch nicht. Sobald der Writer den nächsten `HEARTBEAT` mit Sequenznummer 1 sendet, erkennt der

Abb. 2.4: RTPS Simple Endpoint Discovery: Subscriber zu Publisher



Reader, dass ihm die Nachricht mit Sequenznummer 1 fehlt und fordert diese über ein ACKNACK an. Der Writer sendet daraufhin die fehlende Nachricht erneut via Unicast.

CycloneDDS

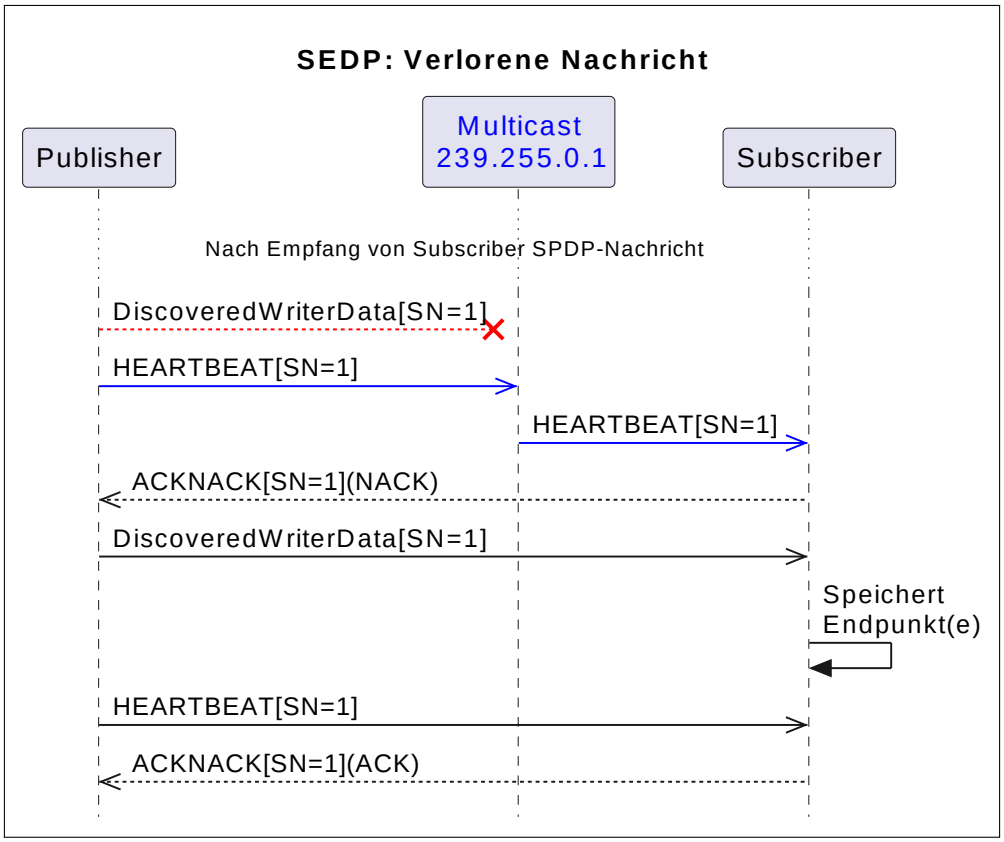
CycloneDDS ist eine quelloffene Implementierung des DDS-Standards, die unter der Eclipse Public License steht. Es wird von der Eclipse Foundation verwaltet und gilt als leichtgewichtige, performante Implementierung des DDS-Standards [5]. CycloneDDS ist eine Standard-Middleware in *Robot Operating System 2* (ROS2) [23], welches im Kontext von ADAS und Autonomem Fahren zum Einsatz kommt.

2.1.2 SOME/IP

SOME/IP[2] ist eine Middleware, die explizit aus dem Automobilbereich stammt und von einem Konsortium verschiedener Hersteller der Branche (AUTOSAR) weiterentwickelt

[2]Die Grundlage für die Erklärungen bildet die SOME/IP-Spezfikation FO 24-11 [2].

Abb. 2.5: RTPS Simple Endpoint Discovery: Verlorene Nachricht



wird. Durch seinen Ursprung ist es auf die Anforderungen in Fahrzeugnetzwerken zugeschnitten und bewusst leichtgewichtig gestaltet. Es weist nativ keine QoS-Fähigkeiten auf und spezifiziert keine eigenen Security-Features [2, S. 86 ff.].

Allgemeine Merkmale

SOME/IP setzt auf ein serviceorientiertes Kommunikationsmodell: Komponenten tauschen Informationen in Form von Services aus, die eindeutig über Service-IDs identifiziert werden. Ein Dienst kann dabei verschiedene Methoden oder Events bereitstellen, die von Clients genutzt oder abonniert werden. Für zeitkritische Übertragungen kommt UDP zum Einsatz, während TCP für zuverlässige Datenübertragung verwendet wird. Die Architektur von SOME/IP ist darauf ausgelegt, auch in großen und verteilten Systemen effizient zu funktionieren. Dank dynamischer Service Discovery können neue Dienste flexibel erkannt und genutzt werden, was die Skalierbarkeit erhöht. Die Verwendung

kompakter Nachrichtenformate soll zudem für einen geringen Overhead und eine hohe Effizienz sorgen. Ein weiterer Aspekt ist die Plattformunabhängigkeit: SOME/IP lässt sich auf unterschiedlichen Betriebssystemen und Hardware-Architekturen einsetzen und ist nativ mit dem AUTOSAR-Stack (Classic und Adaptive) kompatibel.

SOME/IP Service Discovery

Die Service Discovery ist bei SOME/IP in der separaten Spezifikation SOME/IP-SD beschrieben [3]. Der Transport erfolgt ausschließlich über UDP. Während des Discovery-Prozesses in SOME/IP-SD werden Dienste und deren Instanzen im Netzwerk bekannt gemacht und gesucht.

Nach dem Hochfahren durchläuft jeder SOME/IP-Teilnehmer sequenziell drei Phasen, die für jeden angebotenen (Publisher) bzw. abonnierten (Subscriber) Service individuell einen Zustandsautomaten bilden:

Initial Wait Phase

- Startet, wenn ein Service (Publisher oder Subscriber) verfügbar und die Netzwerkschnittstelle bereit ist.
- Es wird eine zufällige Wartezeit (zwischen INITIAL_DELAY_MIN[3] und INITIAL_DELAY_MAX) abgewartet, um Netzwerklastspitzen beim Start zu vermeiden.
- Während dieser Phase werden noch keine SD-Nachrichten gesendet.

Repetition Phase

- SD-Nachrichten (`FindService`, `OfferService`; s. u.) werden mehrfach gesendet, um eine schnelle Synchronisation zwischen Publishern und Subscribern zu ermöglichen.
- Die Wartezeit zwischen den Nachrichten verdoppelt sich nach jedem Sendevorgang (exponentielles Backoff, basierend auf REPETITIONS_BASE_DELAY).
- Die Anzahl der Wiederholungen ist durch REPETITIONS_MAX begrenzt.

[3] Angaben in Großbuchtstaben beschreiben durch die SOME/IP-SD-Spezifikation [3] vorgegebene Konfigurationsparameter.

- Hat ein Subscriber in der Initial Wait Phase bereits ein passendes `OfferService` erhalten, geht er direkt in die Main Phase über, ohne FindService-Nachrichten zu senden.

Main Phase

- Publisher senden `OfferService`-Nachrichten in regelmäßigen Abständen (Intervall definiert durch CYCLIC_OFFER_DELAY).
- Subscriber senden keine `FindService`-Nachrichten mehr, solange sie gültige Offers empfangen.
- In dieser Phase wird der Netzwerkverkehr minimiert, da SD-Nachrichten nur noch zyklisch gesendet werden.

Der Nachrichtenaustausch geschieht durch `Offer-` und `Find`-Nachrichten zum Auffinden von Publishern sowie `Subscribe`-Nachrichten zum Abonnieren von Events beim Publisher. Folgende Nachrichtentypen sind für diese Arbeit relevant:

OfferService: Ein Publisher sendet eine `OfferService`-Nachricht, um seine Verfügbarkeit im Netzwerk bekannt zu geben. Diese Nachricht enthält Informationen wie Service-ID, Instanz-ID und unterstützte Methoden/Events.

FindService: Ein Subscriber sendet eine `FindService`-Nachricht, um nach einem bestimmten Dienst zu suchen. Die Nachricht spezifiziert, welche Services und Instanzen gesucht werden. Diese Nachrichten werden nur zu Beginn der SD gesendet und nur, falls nicht bereits ein passendes `OfferService` empfangen wurde. Sie sind insbesondere von Bedeutung, wenn ein Subscriber nach dem Starten eines Publishers initial einen Dienst sucht.

SubscribeEventgroup: Für eventbasierte Kommunikation sendet ein Subscriber eine `SubscribeEventgroup`-Nachricht, um sich für bestimmte Events eines Publishers zu registrieren.

SubscribeEventgroupAck: Der Publisher bestätigt die Registrierung und signalisiert dem Subscriber, dass dieser ab sofort über die gewünschten Events benachrichtigt wird.

StopSubscribeEventgroup: Mit dieser Nachricht kann ein Subscriber die Registrierung für Events beenden.

Die `OfferService`-Nachrichten werden standardmäßig über Multicast gesendet, sodass alle Teilnehmer im Netzwerk diese empfangen können. Hat ein Publisher zuvor ein `FindService` empfangen, so schickt er ein darauf antwortendes `OfferService` via Unicast. Sobald ein Subscriber ein passendes `OfferService` erhält, kann er eine `SubscribeEventgroup`-Nachricht an den Publisher senden, um sich für die gewünschten Events zu registrieren. Der grundlegende SOME/IP-SD-Nachrichtenaustausch besteht demnach aus einem 3-Wege-Handshake (`OfferService` → `SubscribeEventgroup` → `SubscribeEventgroupAck`).

Abb. 2.6 zeigt den grundlegenden Discovery-Nachrichtenaustausch zwischen einem Subscriber und einem Publisher bei SOME/IP-SD[4]. Ein Publisher schickt eine OfferService-Nachricht an eine vordefinierte Multicast-Adresse und Port. Der Subscriber empfängt diese Nachricht und sendet daraufhin ein SubscribeEventgroup via Unicast an den Publisher. Der Publisher bestätigt den Empfang mit einem SubscribeEventgroupAck.

Abb. 2.6: SOME/IP-SD OfferService-Handshake

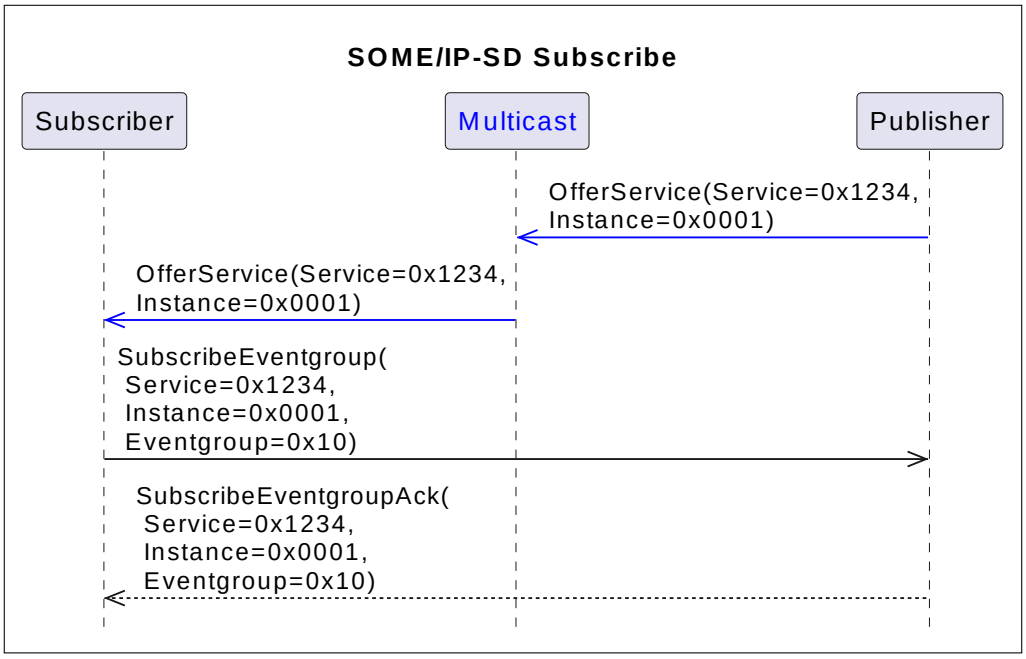
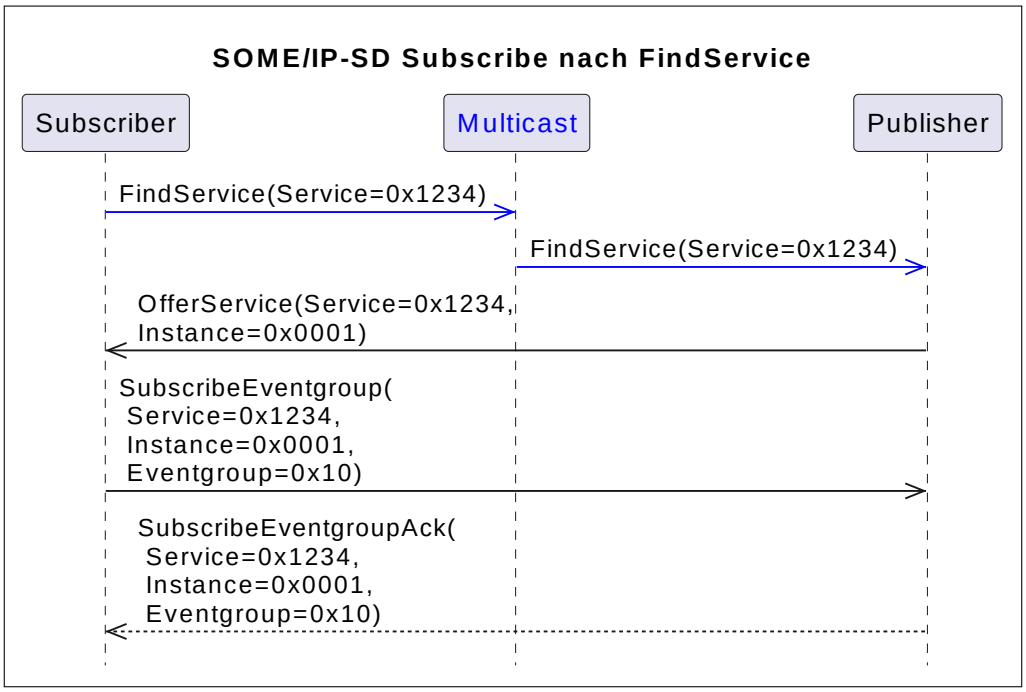


Abb. 2.7 zeigt den Nachrichtenaustausch, wenn ein Subscriber einen Dienst aktiv über eine FindService-Nachricht sucht. In diesem Fall sendet der Subscriber zunächst ein FindService via Multicast. Der Publisher empfängt diese Nachricht und antwortet mit einem

[4]Für die Erstellung der UML-Diagramme in dieser Arbeit wurde, sofern nicht anders angegeben, die Software *PlantUML* [25] verwendet.

OfferService via Unicast. Der Subscriber sendet daraufhin ein SubscribeEventgroup, die der Publisher mit einem SubscribeEventgroupAck bestätigt.

Abb. 2.7: SOME/IP-SD OfferService-Handshake mit FindService



vSomeIP

vSomeIP ist eine quelloffene C++-Implementierung des SOME/IP-Protokolls, die vom Konsortium *Connected Vehicle Systems Alliance* (COVESA) entwickelt und gepflegt wird [8, 9]. Sie wird häufig als Referenzimplementierung in der Automobilindustrie eingesetzt und ist unter der Mozilla Public License (MPL) verfügbar. Konfigurationsparameter werden der Applikation über JSON-Dateien bereitgestellt.

3 Verwandte Arbeiten

3.1 Übersicht

Im folgenden Kapitel werden relevante Arbeiten aus verschiedenen Bereichen vorgestellt, die sich mit SDVs sowie mit Service Discovery in verteilten Systemen, insbesondere im Kontext von SOME/IP und DDS, beschäftigen.

3.2 Automotive Middlewares

Cakir et al. [7] führen eine ausführliche Anforderungsanalyse zu service-orientierten Architekturen im Automobilbereich durch und entwickeln darauf basierend eine Middleware mit einem Multi-Protokoll-Stack, welcher auf den Anwendungszweck zugeschnittene dynamische QoS-Anpassung ermöglicht. SOME/IP wird als IP-basierte Lösung dabei in dieses Middleware-Konzept integriert.

Rumez et al. [26] geben einen Überblick zu hybriden E/E-Architekturen im Automobilbereich und zeigen auf, wie serviceorientierte Systeme mit signalbasierten Systemen verbunden werden können. Darüber hinaus adressieren sie Security-Aspekte im Kontext serviceorientierter Architekturen und stellen eine hybride Beispielarchitektur unter deren Berücksichtigung vor.

Salomon et al. [28] stellen in ihrer Arbeit ein *Domain Name System* (DNS)-basiertes Verfahren zur Authentifizierung und Autorisierung von SOME/IP-Services in Fahrzeugnetzwerken vor. Sie testen ihr Verfahren dabei einerseits mit einer Gefährdungsanalyse und führen andererseits Performance-Messungen mithilfe von Mininet als Netzwerkemulator durch.

Seyler et al. [30] und *Fraccaroli et al.* [10] führen Timing-Analysen der Service Discovery von SOME/IP durch und entwickeln formale Modelle zur Berechnung der Discovery-Latenz unter Berücksichtigung verschiedener Parameter-Konstellationen. Zu DDS/RTPS entwickeln *Sciangula et al.* [29] ebenfalls ein Timing-Modell mit Fokus auf die Implementierung *FastDDS*, jedoch ohne expliziten Fokus auf die Service Discovery.

Parallel zur Entstehung dieser Arbeit erschien von *Klüner et al.* [14] eine Studie, die eine ähnliche Fragestellung behandelt. Die Autoren vergleichen die Middlewares FastDDS, Zenoh und vSomeIP hinsichtlich ihrer Leistungsfähigkeit mithilfe von Kernel-Tracing-Instrumentierung. Es werden die Metriken *Time-to-first-Message* (T_{TFM}), *Communication Time* (T_{Com}), *Scaling Factor* (S) gemessen. Der Aspekt der Service Discovery ist hier implizit in der Metrik T_{TFM} enthalten, umfasst im Gegensatz zur vorliegenden Arbeit jedoch mehr Stack-Delay.

Henle et al. [11] analysieren in ihrer Arbeit, inwiefern ROS2 die speziellen Anforderungen des Automotive-Bereichs erfüllt und ob es eine Alternative zum AUTOSAR-Standard darstellen kann. Die Autoren erstellen eine strukturierte Bewertung anhand verschiedener Anforderungskategorien.

4 Anforderungen an Service Discovery-Protokolle

In diesem Kapitel werden die grundlegenden Anforderungen an die Service Discovery-Protokolle im Kontext von Middlewares dargestellt. Diese Anforderungen bilden die Grundlage für den späteren Vergleich zwischen SOME/IP und DDS in Kapitel 6 und orientieren sich an den spezifischen Gegebenheiten und Herausforderungen im Automobilbereich. Da in dieser Arbeit ausschließlich nicht-funktionale Anforderungen bewertet werden, sind funktionale Anforderungen hier nicht nochmal explizit aufgeführt.

Die nicht-funktionalen Anforderungen definieren Qualitätsattribute und Randbedingungen, die für den Einsatz in sicherheitskritischen Automotive-Systemen von besonderer Bedeutung sind.

Performanz und Determinismus Service Discovery-Protokolle sollten die Echtzeitanforderungen in Automotive-Systemen erfüllen. Dazu zählen eine minimale Latenz bei der Service Discovery und ein geringer Rechenaufwand zur Verarbeitung der Discovery-Nachrichten. Der Austausch der Nachrichten sollte deterministisch innerhalb vorgegebener Zeitfenster geschehen, um Deadlines beim Hochfahren der Fahrzeugsysteme sowie bei der Übertragung kritischer Daten einzuhalten [15].

Ressourceneffizienz und Skalierbarkeit Insbesondere im Kontext von ADAS und Autonomem Fahren wird Bandbreite durch die zu übertragene Datenmenge vieler ECUs zum Flaschenhals [15]. Die begrenzten Hardwareressourcen in *In-Vehicle Network*s (IVNs) erfordern daher eine effiziente Datenübertragung und -verarbeitung. Die Service Discovery-Nachrichten sollten daher Informationen möglichst effizient kodiert übertragen. Darüber hinaus sollte die Middleware im Allgemeinen auch auf kleinen Steuergeräten mit begrenzter Rechenkapazität lauffähig sein, gegebenenfalls mit einem begrenzten Feature-Set [1]. Da eine serviceorientierte Architektur optionale Funktionserweiterungen vorsieht, muss

der Discovery-Mechanismus in der Lage sein, angemessen zu skalieren, d. h. dass eine wachsende Zahl von ECUs im Fahrzeug die Dauer des Discovery-Prozesses nicht überproportional erhöhen sollte [14].

Robustheit und Zuverlässigkeit Das Service Discovery-Protokoll muss robust gegenüber Ausfällen einzelner Komponenten sein und mit temporären Netzwerkunterbrechungen umgehen können. Bei Ausfall eines Teilnehmers oder im Fehlerfall muss das System schnell in der Lage sein, den Ausfall zu erkennen und bei Rückkehr des Teilnehmers die Verbindung schnellstmöglich wieder herstellen können. Daraus resultiert, dass Zustände von Services und Daten zeitlich begrenzte Gültigkeit aufweisen können sollten. Darüber hinaus muss die Discovery verteilt und ohne zentrale Registrierungsinstanz ablaufen können, um bei möglichen Ausfällen einen Single Point of Failure zu verhindern [1].

Konfigurierbarkeit und Anpassungsfähigkeit Die Service Discovery sollte über Konfigurationsparameter an die spezifischen Ressourcen der ECU und des Netzwerks sowie an die Rolle der Applikation anpassbar sein, um den Ablauf und die Geschwindigkeit optimieren zu können [1].

Sicherheit Im Sinne von Security stellen Service Discovery-Mechanismen potenzielle Angriffsvektoren dar, da sie Informationen über verfügbare Dienste im Netzwerk verbreiten. Daher sollten die Protokolle Mechanismen zur Authentifizierung und Verschlüsselung der Kommunikation unterstützen [15, 26]. Insbesondere sollten nur autorisierte ECUs Services entdecken und nutzen können, um unberechtigte Zugriffe zu verhindern. Zudem ist die Integrität der übertragenen Service-Informationen sicherzustellen, um Man-in-the-Middle-Angriffe zu erschweren [26].

5 Versuchsaufbau und Methode

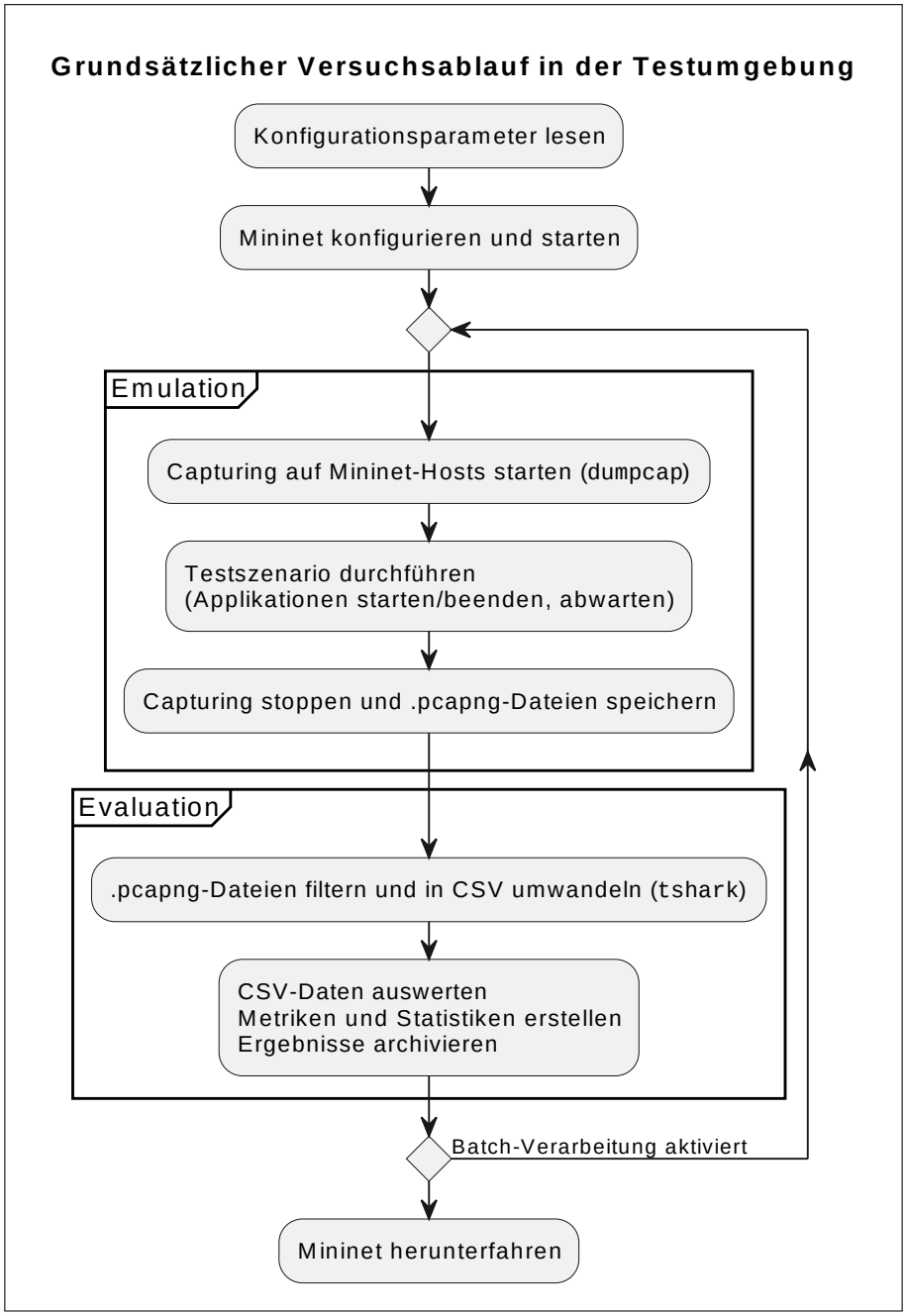
Im folgenden Kapitel werden Versuchsaufbau und Messmethodik für die Evaluation der Service Discovery von DDS und SOME/IP beschrieben. Zuerst wird auf den grundsätzlichen Ablauf und die Implementierung der Testumgebung eingegangen (Abschnitt 5.1). Anschließend wird in Abschnitt 5.2 die Messmethodik für beide Protokolle im Detail erläutert. In 5.3 werden für beide Middlewares die getesteten Konfigurationsparameter vorgestellt sowie in Abschnitt 5.4 die getesteten Teilnehmerkonstellationen im Netzwerk. Abschließend werden in Abschnitt 5.5 Testszenarien beschrieben, welche typische Anwendungsfälle in einem Fahrzeugnetzwerk abbilden sollen.

5.1 Grundsätzlicher Versuchsablauf

In dieser Arbeit werden in einer automatisierten Testumgebung verschiedene Szenarien zur Service Discovery mit DDS und SOME/IP untersucht. In jedem Testlauf wird eine spezifische Teilnehmerkonstellation in einem isolierten Netzwerk simuliert, in dem Publisher und Subscriber dynamisch gestartet und beendet werden können. Der Fokus liegt auf der Messung der Zeit, die benötigt wird, bis alle Teilnehmer sich gegenseitig entdeckt haben. Die Netzwerkumgebung wird mithilfe einer in Mininet emulierten Netzwerkumgebung erstellt.

Abb. 5.1 zeigt den grundsätzlichen Ablauf eines Testlaufs in der Testumgebung. Nach Konfiguration und Hochfahren der Mininet-Umgebung lässt sich die Verarbeitung in die Schritte Emulation und Evaluation einteilen. Je nach Einstellung können Emulation und Evaluation in einer Batch-Verarbeitung mehrfach durchlaufen werden, um verschiedene Konfigurationen automatisiert zu testen.

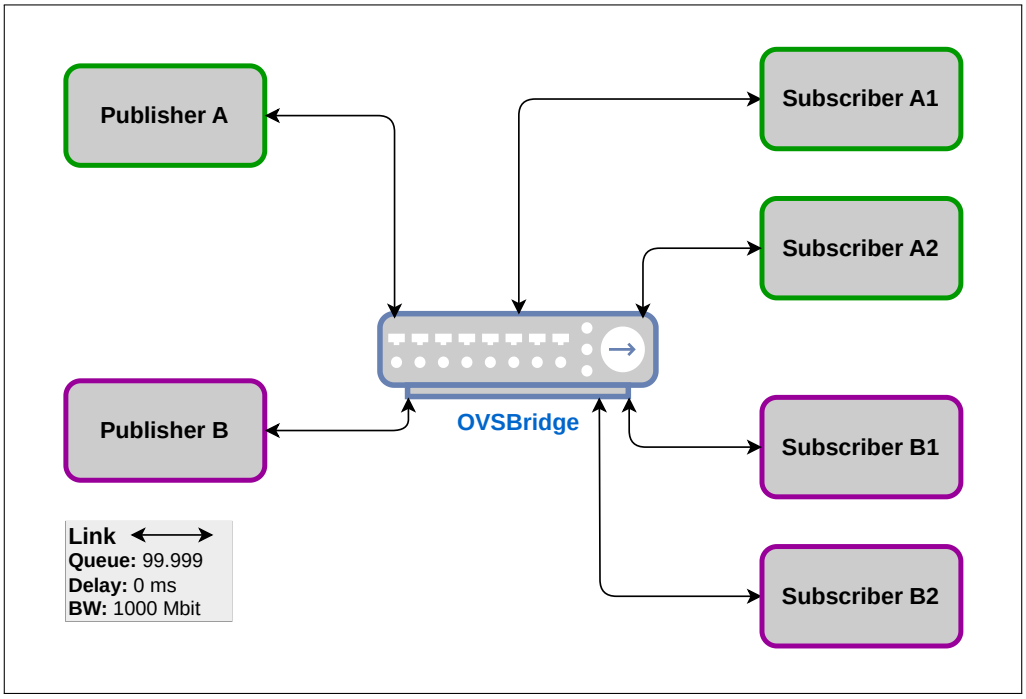
Abb. 5.1: Versuchsablauf in der Testumgebung



5.1.1 Mininet und Netzwerkkonfiguration

Mininet [16, 4] ist ein Netzwerkemulator, der es ermöglicht, virtuelle Netzwerktopologien auf Basis von Netzwerk-Namespaces im Linux-Kernel zu emulieren. Im Rahmen dieser Arbeit wird ein einfaches, flaches Netzwerk mit einem einzigen Switch verwendet, um die Service Discovery-Mechanismen zu vergleichen. Zur einfacheren Vergleichbarkeit bietet jeder Publisher genau eine distinkte Topic (DDS) bzw. ein Event (SOME/IP) an. Jeder Subscriber abonniert genau eine Topic bzw. ein Event. Daraus ergibt sich eine einheitliche 1:N-Beziehung zwischen Publisher und Subscribern und die Zuordnung bei der Analyse der Netzwerkkommunikation wird vereinfacht. Die Topologie in Mininet ist homogen konfiguriert: Den Switch in der sternförmigen Topologie bildet eine virtuelle *Open vSwitch Bridge*. Jeder Teilnehmer läuft auf einem eigenen Host und ist über einen 1000-Mbit-Link, der eine Queue-Größe von 99.999 sowie einen Delay von 0 ms aufweist, mit dem Switch verbunden. Abbildung 5.2 zeigt exemplarisch eine Konstellation mit zwei Publishern, denen jeweils zwei Subscriber zugewiesen sind.

Abb. 5.2: Topologie in Mininet (Beispielkonfiguration)



Die hier genutzte Topologie unterscheidet sich deutlich von der eines Fahrzeugnetzwerks mit zonaler oder Domänenarchitektur. Der Fokus der Analyse in dieser Arbeit liegt auf den für Fahrzeuge typischen Teilnehmerkonstellationen und Laufzeitszenarien.

5.1.2 Implementierung und Durchführung

Für die Messdurchläufe für vSomeIP und CycloneDDS wurde eine Testumgebung[1] aufgebaut, in der die verschiedenen Szenarien automatisiert ausgeführt werden können. Tabelle 5.1 gibt einen Überblick über die verwendete Software und deren Versionen. Die Messungen wurden durchgeführt auf einer virtuellen Maschine mit 60 CPU-Kernen vom Typ *Intel Xeon Gold 6130* (2,1 GHz) und 64 GiB DDR4 RAM.

Für die Durchführung der Messungen wurden minimale C++-Programme geschrieben, die entweder als Publisher oder Subscriber agieren und die Discovery-Protokolle von DDS bzw. SOME/IP implementieren. Die Programme werden automatisiert durch das Python-basierte Testframework gestartet und werden nach Abschluss des Discovery-Prozesses beendet. Die Ausführung der Binaries erfolgt auf den von Mininet erzeugten Hosts. Die Programme erzeugen abseits der Discovery keinen Datenverkehr, um Messungen des Discovery-Netzwerkverkehrs nicht zu verfälschen. In den Szenarien S4 und S5 (siehe Abschnitt 5.5) werden die Anwendungen teils beendet und neugestartet. Dabei werden die Prozesse per SIGKILL beendet, um abrupte Verbindungsabbrüche ohne sauberes Herunterfahren zu simulieren.

5.2 Messzeitpunkte und Vergleichbarkeit

Da die Service Discovery-Protokolle von DDS/RTPS und SOME/IP-SD sich in ihren Abläufen und Nachrichtentypen stark unterscheiden, ist es notwendig, vergleichbare Messzeitpunkte zu definieren, um einen hinreichend objektiven Vergleich zu gewährleisten.

5.2.1 Lokalisierung der Messzeitpunkte

Da die Aufzeichnung des Netzwerkverkehrs auf einer einzelnen Maschine mit Mininet erfolgt, gibt es wenig Abweichung zwischen den Zeitpunkten, zu denen eine Nachricht

[1] Das Repository ist abrufbar unter `https://git.inet.haw-hamburg.de/core/source/someip/dds-vs-someip` (Zugang nur über das VPN der CoRE-Forschungsgruppe).

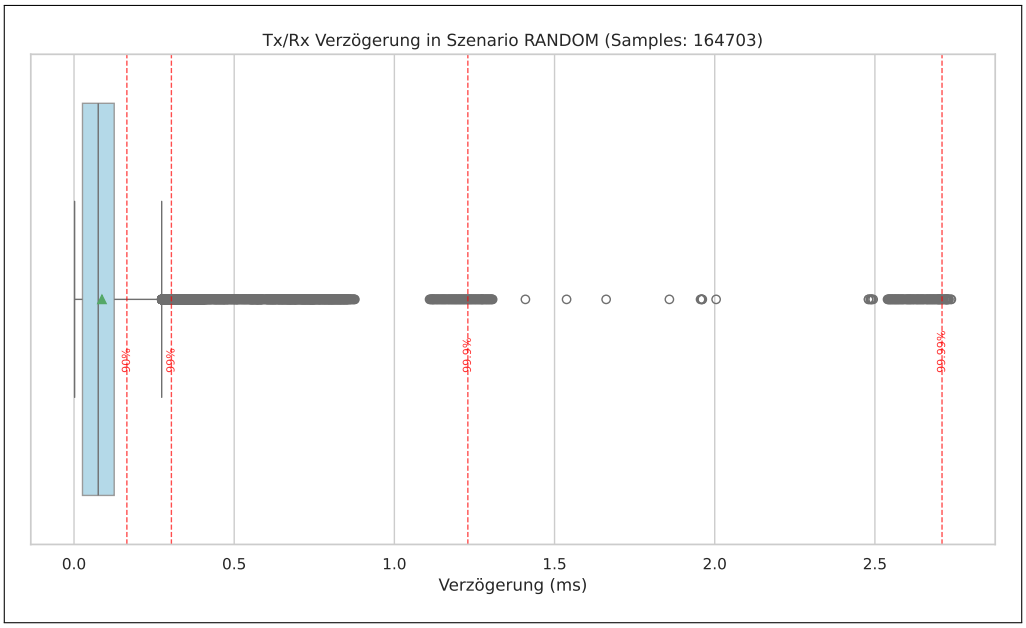
Tab. 5.1: Verwendete Technologien im Testbed

Werkzeug	**Beschreibung**	**Version**
Ubuntu Server	Linux-Distribution mit Mininet-Unterstützung.	22.04
Mininet	Emulator für Netzwerktopologie.	2.3.0d6
Python	Steuerung der Testdurchläufe, Auswertung und Automatisierung.	3.13
C++	Implementierung der Middlewares.	17
CMake	Buildsystem für C++.	3.25
CycloneDDS	Open-Source DDS-Stack.	0.11.0
vSomeIP	Open-Source SOME/IP-Stack.	3.5.5
Loguru (Python)	Python-Logging-Bibliothek.	0.7.3
tshark / dumpcap	Analyse von Netzwerkverkehr (pcap) zur Ermittlung von Discovery-Events.	4.4.7
pytest	Automatisiertes Testen der Python-Komponenten.	8.4.1
Pandas	Datenanalyse und -aufbereitung der Messergebnisse.	2.2.3
statsmodels	Statistische Modelle für Parameterstudie	0.14.4
matplotlib	Visuelle Darstellung der Messergebnisse.	3.10.3
Jupyter Notebooks	Interaktive Auswertung und Visualisierung der Ergebnisse.	7.4.3

von der sendenden Applikation abgeschickt (Tx) und der empfangenden Applikation empfangen (Rx) wird. In Abbildung 5.3 ist die Verteilung der Differenz Rx-Tx für alle Nachrichten in einem Beispiel-Durchlauf dargestellt [2]. Es zeigt sich, dass die Differenz in über 99% aller Fälle weit unter 1 ms liegt, was sich mit den Ergebnissen von *Muelas et al.* [17] deckt. Dort wurde in Experimenten mit 1000 Hosts in Mininet Round-Trip-Time mit einem Median von unter 1 ms gemessen. Aus diesem Grund wird in dieser Arbeit durchgängig der Zeitpunkt des Empfangs (Rx) als Referenz verwendet, da dies die Analyse vereinfacht: Multicast-Nachrichten könnten auf Senderseite sonst teils nicht oder nur durch hohen Aufwand einem Empfänger zugeordnet werden.

[2] Für die Erstellung der Statistik-Diagramme in dieser Arbeit wurde, sofern nicht anders angegeben, die Bibliothek *matplotlib* [12] verwendet.

Abb. 5.3: Differenz Messzeitpunkte Rx–Tx



Es ist zu erwähnen, dass die Evaluation durch Verfahren außerhalb der Netzwerkemulation auf einem einzelnen Host nur begrenzt einsetzbar ist: Wird die Testumgebung außerhalb von Mininet auf mehreren Maschinen ausgeführt, können durch Netzwerklatenzen größere Abweichungen auftreten.

5.2.2 Auswahl der Messzeitpunkte

Da die Art und der Ablauf des Nachrichtenaustauschs bei RTPS und SOME/IP-SD unterschiedlich sind, müssen für einen ordnungsgemäßen Vergleich vergleichbare Messzeitpunkte definiert werden. In Tabelle 5.2 sind die äquivalenten Ereignisse für beide Protokolle auf Publisher- und Subscriberseite dargestellt[3].

Während der Nachrichtenaustausch bei RTPS Service Discovery „symmetrisch“ ist, da sowohl Publisher als auch Subscriber gleichförmige Nachrichten (SPDP u. SEDP) schicken, ist der Ablauf bei SOME/IP-SD asymmetrisch: Publisher senden *OfferService*- und *SubscribeEventgroupAck*-Nachrichten, während Subscriber *FindService*- und *SubscribeEventgroup*-Nachrichten senden. Bei SOME/IP ist zu beachten, dass *FindService*-

[3] Angaben in Klammern bedeuten in Tabelle 5.2, dass das Ereignis zwar registriert wird, aber für die Messung nicht relevant ist.

	DDS/RTPS		SOME/IP-SD	
Ereignis	**Subscriber**	**Publisher**	**Subscriber**	**Publisher**
Teilnehmer T entdeckt	SPDP-Nachricht von T empfangen	SPDP-Nachricht von T empfangen	*FindService* o. *OfferService* von T empfangen	*FindService* o. *SubscribeEventgroup* o. *OfferService* von T empfangen
Subscriber S entdeckt	(SEDP-Nachricht von S empfangen)	SEDP-Nachricht von S empfangen	(*FindService* von S empfangen)	*FindService* o. *SubscribeEventgroup* von P empfangen
Publisher P entdeckt	SEDP-Nachricht von P empfangen	(SEDP-Nachricht von P empfangen)	*OfferService* von P empfangen	(*OfferService* von P empfangen)
Discovery abgeschlossen	SPDP/SEDP-Nachrichten vom Publisher empfangen	SPDP/SEDP-Nachrichten von allen Subscribern empfangen	*SubscribeEventgroupAck* vom Publisher empfangen	*Subscribe-Eventgroup* von allen Subscribern empfangen

Tab. 5.2: Äquivalenzen von Ereignissen bei RTPS und SOME/IP-SD

Nachrichten nicht zwangsläufig gesendet werden (siehe Abschnitt 2.1.2). Daher kann das Ereignis „Subscriber entdeckt“ auf Publisherseite entweder durch den Empfang einer *FindService*-Nachricht oder durch den Empfang einer *SubscribeEventgroup*-Nachricht geschehen.

Die Messungen der Discovery-Dauer werden auf mehreren Ebenen durchgeführt:

- Für jeden Publisher individuell
- Für jeden Subscriber individuell
- Für die Gesamtheit aller Teilnehmer

Hierbei wird jeweils die Zeitspanne zwischen erster relevanter Nachricht und letzter relevanter Nachricht gemessen. Relevant bedeutet in diesem Kontext: explizit für die vollständige Discovery vorgesehen und benötigt. In den folgenden Abschnitten wird die Messmethodik für beide Protokolle im Detail erläutert.

Notation in Formeln und Diagrammen

Im Folgenden wird für die Definition der Messzeitpunkte und Latenzberechnungen in Sequenzdiagrammen und Formeln die folgende Notation verwendet:

$\mathcal{T}$ Die Menge aller Teilnehmer des Testlaufs.

$\mathcal{S}$ Die Menge aller Subscriber des Testlaufs.

$\mathcal{S}(p)$ Menge aller Subscriber des Testlaufs, die Publisher p zugeordnet sind.

$\mathcal{P}$ Die Menge aller Publisher des Testlaufs.

$t^{(x)}_{y,\langle \texttt{msg} \rangle}$ Zeitpunkt, zu dem Teilnehmer x die erste Nachricht vom Typ `msg` von Teilnehmer y empfängt.

$t^{(p)}_{\text{start}}$ Frühester relevanter Messzeitpunkt für Publisher p.

$\Delta t^{(x)}_{\text{SD}}$ Gesamtdauer des Discovery-Prozesses für Teilnehmer x (die Zeitspanne, in der x alle seine zugewiesenen Kommunikationspartner gefunden hat).

$\Delta t^{(p)}_{s,\text{SD}}$ Dauer des Discovery-Prozesses für Publisher p mit Subscriber s (die Zeitspanne, in der Publisher p Subscriber s vollständig entdeckt hat).

Δt_{total} Gesamtdauer des Discovery-Prozesses über alle Teilnehmer hinweg.

5.2.3 Messmethodik bei DDS/RTPS

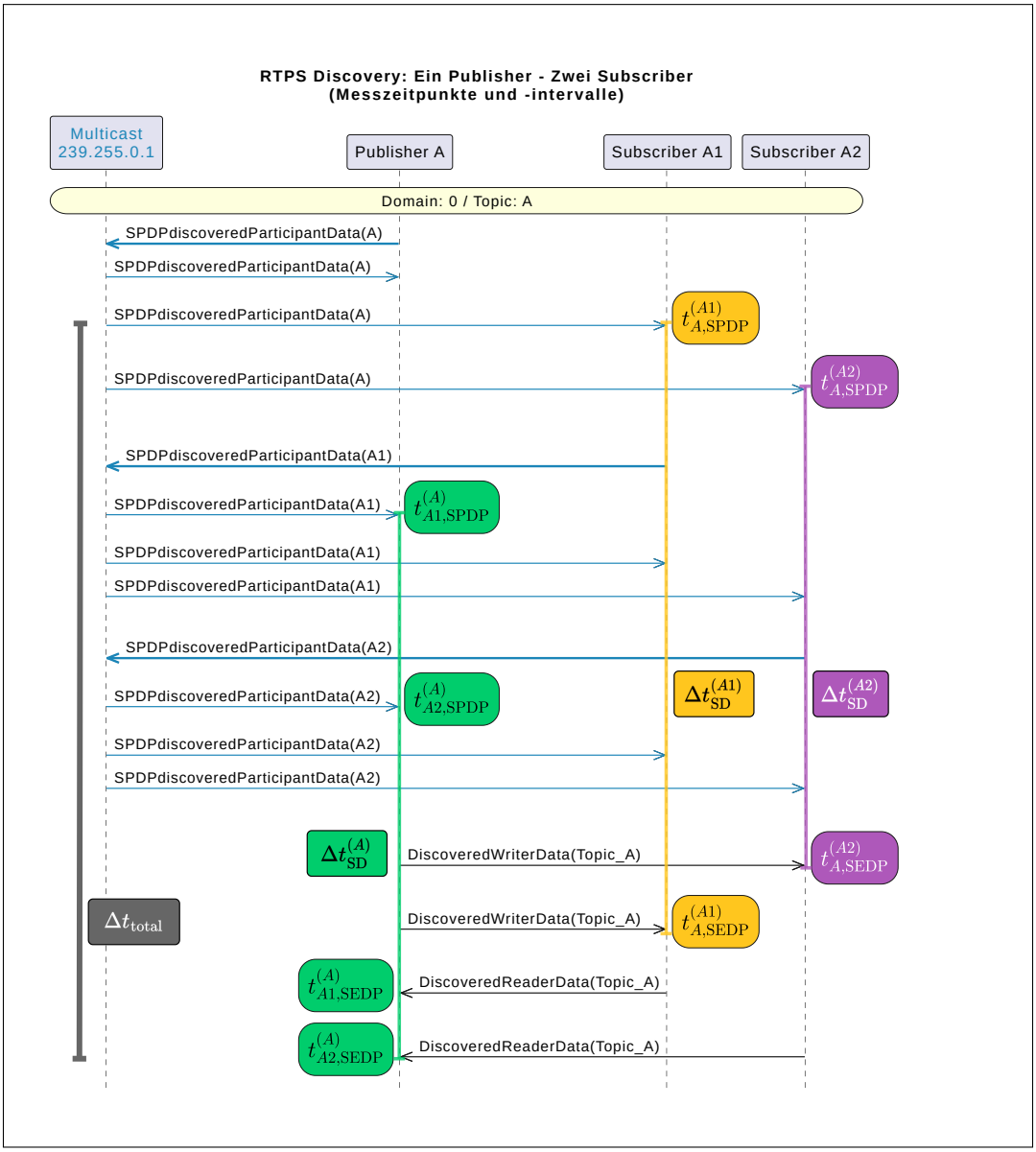
Bezogen auf die Zeitmessung gibt es zwei kritische Zeitpunkte:

1. der Moment, in dem jeder Teilnehmer alle SPDP-Nachrichten empfangen hat
2. der Moment, wenn alle SEDP-Nachrichten empfangen und somit alle Endpunkte bekannt sind.

Bei RTPS beginnt die Discovery mit dem Versand von SPDP-Nachrichten unmittelbar nach dem Start eines Participants. Die Discovery gilt als abgeschlossen, sobald alle relevanten Endpunkte (Publisher/Subscriber) einander bekannt sind und die entsprechenden SEDP-Nachrichten ausgetauscht wurden.

Abbildung 5.4 zeigt die erfassten Messzeitpunkte im RTPS-Discovery-Prozess. Publisher A hat zwei Subscriber $A1$ und $A2$, die Topic A bei ihm abonnieren wollen. Er erfasst

Abb. 5.4: RTPS Discovery: Messzeitpunkte



für beide Subscriber jeweils einen Zeitstempel für den Erhalt ihrer SPDP-Nachrichten ($t^{(A)}_{A1,\mathrm{SPDP}}$ und $t^{(A)}_{A2,\mathrm{SPDP}}$). Anschließend erhält Publisher A zwei SEDP-Nachrichten, für die er ebenfalls zwei Zeitstempel ($t^{(A)}_{A1,\mathrm{SEDP}}$ und $t^{(A)}_{A2,\mathrm{SEDP}}$) erfasst. Für Publisher A lassen sich nun mehrere Latenzen bestimmen: Die einzelnen Latenzen für die individuelle Kommunikation mit den jeweiligen Subscribern (Formeln (5.1) u. (5.2)) sowie die Publisher-Gesamtlatenz (5.3), welche vom Zeitpunkt der ersten empfangenen, benötigten SPDP-Nachricht bis zum Zeitpunkt der letzten benötigten SEDP-Nachricht reicht.

$$\Delta t^{(A)}_{A1,\mathrm{SD}} = t^{(A)}_{A1,\mathrm{SEDP}} \; - \; t^{(A)}_{A1,\mathrm{SPDP}} \tag{5.1}$$

$$\Delta t^{(A)}_{A2,\mathrm{SD}} = t^{(A)}_{A2,\mathrm{SEDP}} \; - \; t^{(A)}_{A2,\mathrm{SPDP}} \tag{5.2}$$

$$\Delta t^{(A)}_{\mathrm{SD}} = \max_{s \in \mathcal{S}(A)} t^{(A)}_{s,\mathrm{SEDP}} \; - \; \min_{s \in \mathcal{S}(A)} t^{(A)}_{s,\mathrm{SPDP}} \tag{5.3}$$

Erstere sind aus Platzgründen nicht auf dem Sequenzdiagramm eingezeichnet. Auf Seiten der beiden Subscriber lässt sich jeweils individuell ermitteln, wann diese die SPDP- und SEDP-Nachrichten des Publishers erhalten haben. Daraus lassen sich jeweils die Subscriber-Latenzen (Formeln (5.4) u. (5.5)) bestimmen.

$$\Delta t^{(A1)}_{\mathrm{SD}} = t^{(A1)}_{A,\mathrm{SEDP}} \; - \; t^{(A1)}_{A,\mathrm{SPDP}} \tag{5.4}$$

$$\Delta t^{(A2)}_{\mathrm{SD}} = t^{(A2)}_{A,\mathrm{SEDP}} \; - \; t^{(A2)}_{A,\mathrm{SPDP}} \tag{5.5}$$

Aus allen erfassten Zeitpunkten lässt sich zudem die System-Gesamtlatenz (5.6) des Discovery-Prozesses über alle Teilnehmer (alle Publisher und alle Subscriber) hinweg berechnen. Hierzu wird der früheste Zeitpunkt einer empfangenen SPDP-Nachricht eines Teilnehmers vom spätesten Zeitpunkt einer empfangenen SEDP-Nachricht eines Teilnehmers subtrahiert.

$$\Delta t_{\mathrm{total}} = \max_{x \in \mathcal{T}} t^{(x)}_{\mathrm{SEDP}} \; - \; \min_{x \in \mathcal{T}} t^{(x)}_{\mathrm{SPDP}} \tag{5.6}$$

5.2.4 Messmethodik bei SOME/IP-SD

Bei SOME/IP-SD startet der Discovery-Prozess mit dem Senden eines OfferService durch den Publisher oder eines FindService durch den Subscriber. Die Discovery gilt als abgeschlossen, wenn der letzte Subscriber ein SubscribeEventgroupAck vom Publisher empfangen hat.

Wie in Abschnitt 2.1.2 beschrieben, ist das Senden eines FindService bei SOME/IP-SD optional. Dies wirkt sich auf die Messung der Latenzen aus, da der Beginn der Messung auf Publisherseite je nach Auftreten entweder mit dem ersten FindService oder dem ersten SubscribeEventgroup beginnt. Abbildungen 5.5 und 5.6 zeigen die Messmethodiken für beide Varianten. In beiden Beispielen gibt es einen Publisher A, der Service `0x1234` anbietet, sowie zwei Subscriber $A1$ und $A2$, die Eventgroup `0x001` bei ihm abonnieren wollen. In den nachfolgenden Formeln werden diese Teilnehmer beispielhaft referenziert.

In dem Fall, dass Publisher A kein FindService von einem seiner Subscriber empfängt (Abb. 5.5), beginnt die Messung mit dem Empfang des ersten SubscribeEventgroup eines Subscribers ($t^{(A)}_{A1,\mathrm{Sub}}$ und $t^{(A)}_{A2,\mathrm{Sub}}$).

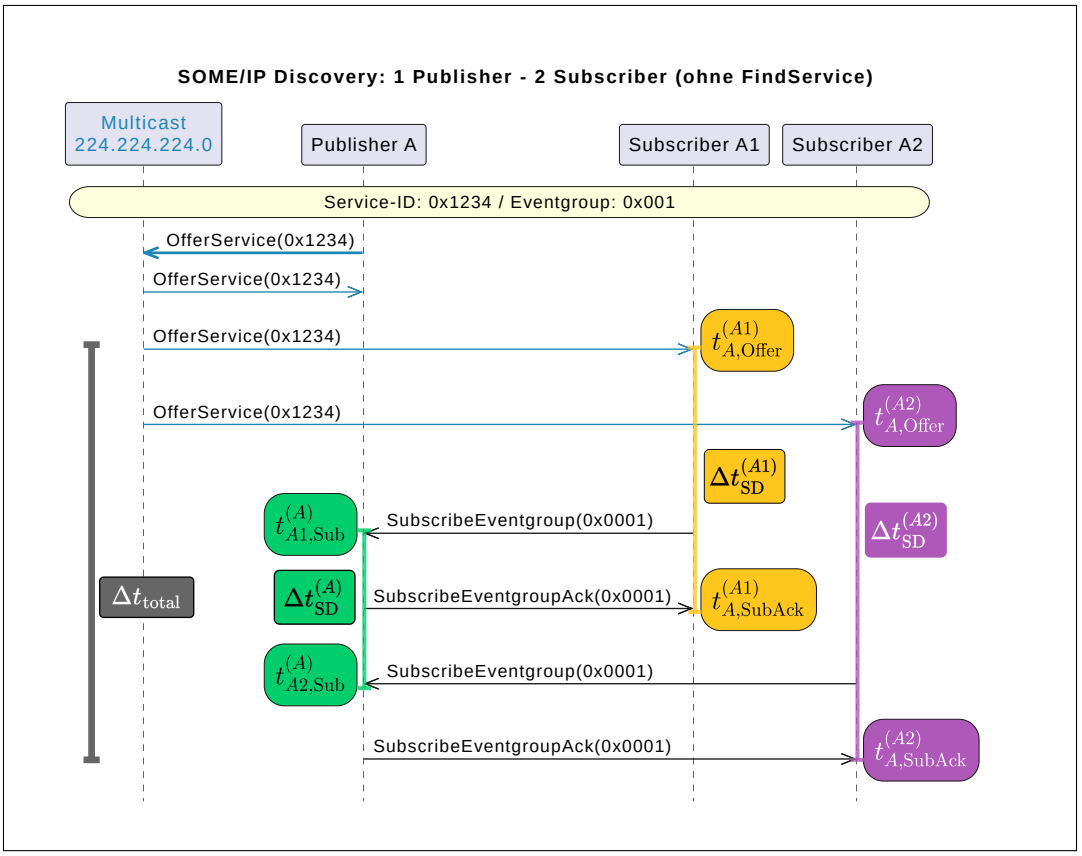
Empfängt Publisher A hingegen zu Beginn der Discovery-Phase ein oder mehrere FindService vor dem ersten SubscribeEventgroup ($t^{(A)}_{A1,\mathrm{Find}}$; in Abb. 5.6), so errechnet sich die Publisher-Latenz $t^{(A)}_{\Delta,\mathrm{SD}}$ aus der letzten benötigten SubscribeEventgroup-Nachricht und der ersten empfangenen FindService-Nachricht. Formel (5.7) beschreibt beide Fälle.

$$\Delta t^{(A)}_{\mathrm{SD}} = \max_{s \in \mathcal{S}(A)} t^{(A)}_{s,\mathrm{Sub}} \; - \; \min\left(\left\{t^{(A)}_{s,\mathrm{Find}} \mid s \in \mathcal{S}(A)\right\} \cup \left\{t^{(A)}_{s,\mathrm{Sub}} \mid s \in \mathcal{S}(A)\right\}\right) \tag{5.7}$$

Auf Subscriberseite werden jeweils Zeitstempel für das erste OfferService von Publisher A registriert ($t^{(A1)}_{A,\mathrm{Offer}}$ u. $t^{(A2)}_{A,\mathrm{Offer}}$). Nach dem Senden der SubscribeEventgroup-Nachrichten werden anschließend jeweils Zeitstempel für den Empfang der ersten SubscribeEventgroupAck-Nachrichten von Publisher A registriert ($t^{(A1)}_{A,\mathrm{SubAck}}$ u. $t^{(A2)}_{A,\mathrm{SubAck}}$). Die individuellen Subscriber-Latenzen ((5.8) u. (5.9)) errechnen sich aus den jeweiligen Differenzen zwischen SubscribeEventgroupAck- und OfferService-Nachrichten.

$$\Delta t^{(A1)}_{\mathrm{SD}} = t^{(A1)}_{A,\mathrm{SubAck}} \; - \; t^{(A1)}_{A,\mathrm{Offer}} \tag{5.8}$$

Abb. 5.5: SOME/IP-SD: Messzeitpunkte ohne FindService



$$\Delta t^{(A2)}_{\mathrm{SD}} = t^{(A2)}_{A,\mathrm{SubAck}} - t^{(A2)}_{A,\mathrm{Offer}} \tag{5.9}$$

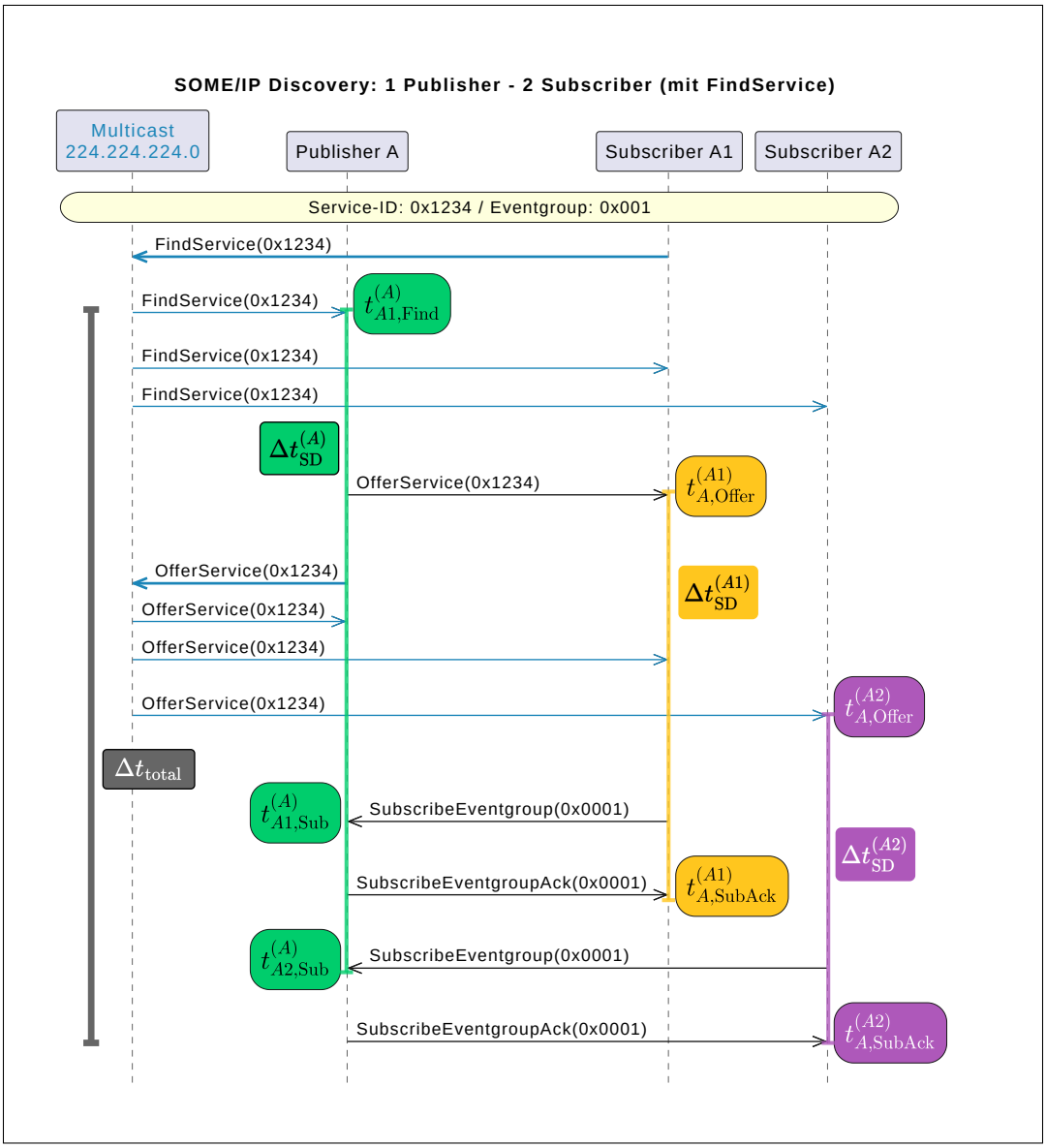
Die Subscriber-Gesamtlatenz (5.10) betrachtet alle Subscriber, vom ersten Empfang einer OfferService-Nachricht bis zur letzten SubscribeEventgroupAck-Nachricht.

$$\Delta t_{\mathrm{Subs}} = \max_{s \in \mathcal{S}} t^{(s)}_{\mathrm{SubAck}} - \min_{s \in \mathcal{S}} t^{(s)}_{\mathrm{Offer}} \tag{5.10}$$

Für die Publisher-Gesamtlatenz (i.e. über alle Publisher hinweg) wird zunächst die früheste relevante Nachricht je Publisher definiert (5.11). Diese berücksichtigt sowohl FindService- als auch SubscribeEventgroup-Nachrichten.

$$t^{(p)}_{\mathrm{start}} = \min\left(\left\{t^{(p)}_{s,\mathrm{Find}} \mid s \in \mathcal{S}(p)\right\} \cup \left\{t^{(p)}_{s,\mathrm{Sub}} \mid s \in \mathcal{S}(p)\right\}\right) \tag{5.11}$$

Abb. 5.6: SOME/IP-SD: Messzeitpunkte mit FindService



Die Publisher-Gesamtlatenz (5.12) berechnet sich über alle Publisher hinweg, vom frühesten Startzeitpunkt eines Publishers bis zur letzten empfangenen SubscribeEventgroup-Nachricht.

$$\Delta t_{\mathrm{Pubs}} = \max_{p \in \mathcal{P}} \left(\max_{s \in \mathcal{S}(p)} t^{(p)}_{s,\mathrm{Sub}} \right) - \min_{p \in \mathcal{P}} t^{(p)}_{\mathrm{start}} \tag{5.12}$$

Die System-Gesamtlatenz (5.13) errechnet sich aus der Differenz der letzten empfangenen, relevanten Nachricht und der ersten relevanten Nachricht über alle Teilnehmer hinweg.

$$\Delta t_{\text{total}} = \max_{s \in \mathcal{S}} t^{(s)}_{\text{SubAck}} \; - \; \min_{x,y \in \mathcal{T}} t^{(x)}_{y,\text{m}} \quad \text{mit m} \in \{\text{Find}, \text{Offer}\} \tag{5.13}$$

5.2.5 Protokolleffizienz

Neben den Latenzen lassen sich auch die übertragenen Datenmengen im Discovery-Prozess vergleichen. Hierzu wird die Gesamtzahl der im Discovery-Zeitraum gesendeten Bytes (also Nachrichtengrößen) betrachtet und aufsummiert. Die Analyse erfolgt für die Gesamtheit aller Teilnehmer. Formel (5.14) beschreibt die entsprechende Berechnung. Die Nachrichtengröße wird immer auf Seite des Senders gemessen, damit Multicast-Nachrichten nicht fälschlicherweise mehrfach gezählt werden. Hierbei ist zu beachten, dass ausschließlich die Nachrichten berücksichtigt werden, die für die Service Discovery zwingend notwendig sind. Bei DDS werden nur SPDP- und SEDP-Nachrichten betrachtet, während bei SOME/IP-SD nur FindService-, OfferService-, SubscribeEventgroup- und SubscribeEventgroupAck-Nachrichten einbezogen werden.

Legende

B_{total} Gesamtzahl der gesendeten Bytes im Discovery-Zeitraum.

$\mathcal{D}$ Menge aller relevanten Discovery-Nachrichten zwischen dem ersten und letzten relevanten Ereignis.

b_i Größe (in Bytes) der i-ten Discovery-Nachricht in $\mathcal{D}$.

$$B_{\text{total}} = \sum_{i \in \mathcal{D}} b_i \tag{5.14}$$

5.3 Konfigurationsparameter

Die zuvor genannten Messungen werden unter Einfluss verschiedener Konfigurationsparameter durchgeführt.

Teilweise haben sie (in)direkte Entsprechungen, teilweise sind sie nur in einer der beiden Implementierungen vorhanden. Die einzelnen Parameterwerte sind für vSomeIP in Tabelle 5.4 und für CycloneDDS in Tabelle 5.3 aufgeführt.

Die Auswahl der getesteten Parameterwerte orientiert sich an den Voreinstellungen der jeweiligen Implementierung sowie an gezielten Abweichungen, um deren Einfluss auf die Service Discovery zu untersuchen. Während der Testläufe werden sowohl die Standardkonfigurationen als auch ausgewählte Parameterkombinationen verwendet. Nicht alle Parameter sind sinnvoll miteinander kombinierbar, daher beschränkt sich die Untersuchung auf eine gezielte Auswahl sinnvoller Kombinationen.

Tabelle 5.3 zeigt die in CycloneDDS verwendeten Konfigurationsparameter.

Der Wert von `HeartbeatInterval` gibt das Zeitintervall an, in dem Heartbeat-Nachrichten zwischen Publishern und Subscribern ausgetauscht werden, um den Übertragungsstatus der Nachrichten zu überwachen und die Erreichbarkeit der Endpunkte zu verifizieren. Es ist Bestandteil der verlässlichen Kommunikation in DDS und damit auch der SEDP-Phase. Es ist relevant, da es einerseits die Menge an Netzwerkverkehr massiv beeinflusst und andererseits hilft, Paketverluste zu detektieren (was in der hiesigen Mininet-Emulation jedoch zweitrangig ist).

`LeaseDuration` gibt die Standard-Gültigkeits-Dauer für DDS-Participants an. Nach Ablauf dieser Zeit ohne erneute SPDP-Nachricht wird der Participant als nicht mehr verfügbar betrachtet. Dieser Parameter ist insbesondere in den Szenarien 4 und 5 von Bedeutung, in denen Teilnehmer temporär ausfallen. Die Ausfallzeit wird in diesen Szenarien so gewählt, dass sie einmal länger ist als die konfigurierte `LeaseDuration` und einmal so, dass sie kürzer ist.

`SPDPInterval` beschreibt das Intervall, in dem periodische SPDP-Nachrichten gesendet werden. Da sie den Lease eines Participants erneuern, müssen sie in einem Intervall $<$ LeaseDuration gesendet werden. In der Voreinstellung bei CycloneDDS beträgt das Intervall automatisch ca. 80% der `LeaseDuration`. Es muss kürzer als die LeaseDuration sein, da sonst Teilnehmer als nicht mehr verfügbar betrachtet werden könnten, obwohl sie noch aktiv sind. In den Testläufen wird das `SPDPInterval` implizit durch die Wahl der LeaseDuration gesetzt (10 Sekunden LeaseDuration entsprechen also ca. 8 Sekunden `SPDPInterval`). Ähnlich wie beim Heartbeat-Intervall beeinflusst dieser Parameter sowohl die Netzwerklast als auch die Schnelligkeit der (Re)Discovery.

Mithilfe des Parameters `SPDPResponseMaxDelay` kann eine Obergrenze für ein Verzögerungsintervall für SPDP-Antworten konfiguriert werden. Die Teilnehmer warten nach dem Empfang einer SPDP-Nachricht eine zufällige Zeit innerhalb dieses Intervalls, bevor sie auf diese antworten. Dies soll Lastspitzen im Netzwerk vermeiden, wenn viele Teilnehmer antworten. Neben dem Standardwert von 0 ms wird in den Testszenarien auch ein Wert von 1000 ms verwendet, um den Einfluss auf die Discovery-Zeiten zu untersuchen. 1000 ms ist der höchste einstellbare Wert. Dies ist nicht in der offiziellen Dokumentation beschrieben, ist jedoch im Quellcode von CycloneDDS ersichtlich[4] und lies sich durch Tests bestätigen.

`DiscoveredLocatorPruneDelay` gibt an, wie lange an entdeckte (Unicast-)Locators nach Ablauf der LeaseDuration noch Ping-Nachrichten gesendet werden, bevor sie verworfen werden. In der Voreinstellung sind dies 60 Sekunden. In den Testszenarien wird dieser Wert auch mit 0s getestet, um ein sofortiges Entfernen von Locators nach Lease-Ablauf zu erzwingen. Auch dieser Wert könnte Auswirkungen auf den Netzwerkverkehr und die Geschwindigkeit der Rediscovery haben, da ein Delay von 0 Sekunden Locators umgehend verwirft und diese anschließend neu entdeckt werden mpssen.

Parameter	**Beschreibung**	**Voreinstellung**	**Getestete Werte**
`HeartbeatInterval`	Zeitintervall zwischen Heartbeats	100ms	1s, 5s
`LeaseDuration`	Gültigkeitsdauer für Participants (bestimmt SPDPInterval)	10s (→ SPDPInterval 8s)	30s (→ SPDPInterval 24s)
`SPDPResponseMaxDelay`	Maximale Verzögerung für SPDP-Antworten	0s	1s
`DiscoveredLocator-PruneDelay`	Zeit, die entdeckte (Unicast) Locators nach Lease-Ablauf noch angepingt werden, bevor sie verworfen werden	60s	0s

Tab. 5.3: Ausgewählte Parameterwerte in CycloneDDS

Tabelle 5.4 zeigt die in vSomeIP verwendeten Konfigurationsparameter.

`cyclic_offer_delay` beschreibt die Anzahl Millisekunden, die zwischen den periodischen OfferService-Nachrichten eines Services vergehen. Der Parameter ist relevant, da er sowohl die Netzwerkauslastung als auch die Schnelligkeit der Discovery beeinflusst: Kür-

[4] `https://github.com/eclipse-cyclonedds/cyclonedds/blob/master/src/core/ddsi/src/ddsi_discovery_spdp.c`; abgerufen am 03.11.2025

zere Intervalle führen zu mehr Netzwerkverkehr, ermöglichen aber auch eine schnellere (Re)Discovery von Services nach Ausfällen.

`initial_delay_min` und `initial_delay_ max` beschreiben die minimale und maximale Dauer, die ein Teilnehmer zu Beginn in der *Initial Wait Phase* bleibt. Der Wert wird durch die Software innerhalb der beiden Grenzen zufällig gewählt. Diese Parameter sind relevant, da sie die Startzeiten der Teilnehmer verteilen und somit Lastspitzen im Netzwerk zu Beginn der Discovery-Phase reduzieren können.

Der `ttl`-Wert gibt die Gültigkeitsdauer eines Services in Sekunden an. Nach Ablauf dieser Zeit wird der Service als nicht mehr verfügbar betrachtet, sofern keine erneute OfferService-Nachricht empfangen wurde. Dieser Parameter ist insbesondere in den Szenarien 4 und 5 von Bedeutung, in denen Teilnehmer temporär ausfallen. Die Ausfallzeit wird einmal so gewählt, dass sie länger ist als die konfigurierte `ttl` und einmal so, dass sie kürzer ist. Standardmäßig ist dieser Wert in vSomeIP so gesetzt, dass Services bis zum nächsten Neustart des Teilnehmers gültig bleiben.

Parameter	**Beschreibung**	**Voreinstellung**	**Getestete Werte**
`cyclic_offer_delay`	Zeitintervall zwischen periodischen OfferService-Nachrichten	1000ms	500ms, 5000ms
`initial_delay_{min, max}`	Mindest- und Maximalverzögerung vor erster OfferService-Nachricht	(0ms, 3000ms)	(0ms, 0ms), (0ms, 500ms), (500ms, 2000ms)
`ttl`	Gültigkeitsdauer eines Services	0xFFFFFF (bis zum nächsten Neustart)	5s (einmal Ausfallzeit länger, einmal Ausfallzeit kürzer)

Tab. 5.4: Ausgewählte Parameterwerte in vSomeIP

Tabelle 5.5 stellt die ausgewählten Konfigurationsparameter aus vSomeIP und CycloneDDS nochmals gegenüber und zeigt, welche Entsprechungen und Abweichungen zwischen den beiden Implementierungen bestehen.

5.4 Konstellation der Netzwerkteilnehmer

Die Anzahl der Netzwerkteilnehmer und das Verhältnis aus Publishern und Subscribern wird in den verschiedenen Szenarien variiert, um die Skalierbarkeit und Effizienz der Ser-

Funktion	**vSomeIP Parameter**	**CycloneDDS Parameter**
Intervall periodischer Discovery-Nachrichten	`cyclic_offer_delay`	`SPDPInterval`
Gültigkeitsdauer eines Service/Teilnehmers	`ttl`	`LeaseDuration`
Initiale Startverzögerung vor erster Discovery-Nachricht	`initial_delay_min`, `initial_delay_max`	(nicht verfügbar)
Intervall periodischer Heartbeat-Nachrichten	(nicht verfügbar)	`HeartbeatInterval`
Verzögerung bei SPDP-Antworten	(nicht verfügbar)	`SPDPResponseMaxDelay`
Anpingen nach Ablauf der Gültigkeitsdauer	(nicht verfügbar)	`DiscoveredLocator-PruneDelay`

Tab. 5.5: Gegenüberstellung ausgewählter Discovery-Konfigurationsparameter in vSomeIP und CycloneDDS

vice Discovery-Mechanismen unter unterschiedlichen Lastbedingungen zu untersuchen. Moderne Fahrzeuge enthalten bis zu 150 ECUs, mit steigender Tendenz [26]. Salomon et al. [27] geben ein Verhältnis von durschnittlich 2,1 Subscribern pro Publisher an, sowie ein Intervall von min. 1 bis max. 4 Subscribern pro Publisher. Daran angelehnt werden in dieser Arbeit fünf Teilnehmerkonstellationen getestet, welche in Tabelle 5.6 dargestellt sind.

Konstellation	**Publisher**	**Subscriber je P.**	**Insgesamt**
K1	10	5	60
K2	25	4	125
K3	50	4	250
K4	100	3	400
K5	200	2	600

Tab. 5.6: Anzahl der Netzwerkteilnehmer je Konstellation

Die Konstellationen sollen zeigen, inwiefern die Middlewares mit steigendem Netzwerkverkehr und erhöhter Rechenlast skalieren können. Das Verhältnis von Publishern zu Subscribern orientiert sich an realistischen Anwendungsfällen in Fahrzeugnetzwerken. Konstellation K2 wurde nachträglich zwischen K1 und K3 eingeführt, da die Ergebnisse im Bereich K3 in den DDS-Testläufen Performance-Einbrüche zeigten. Dies wird in Kapitel 6 näher erläutert.

5.5 Szenarien

Um die Service Discovery-Mechanismen von CycloneDDS und vSomeIP unter verschiedenen Bedingungen zu vergleichen, wurden fünf unterschiedliche Szenarien entworfen. Diese Szenarien simulieren typische Anwendungsfälle in einem Fahrzeugnetzwerk und ermöglichen die Untersuchung der Discovery-Dauer und -Zuverlässigkeit unter variierenden Startbedingungen und Netzwerkereignissen. Tabelle 5.7 bietet eine Übersicht über die einzelnen Szenarien, deren Ablauf, Zielsetzung sowie die relevanten Konfigurationsparameter je Implementierung.

Szenario- und plattformübergreifend werden – abgesehen vom Discovery-Netzwerkverkehr – keine Daten zu Topics (DDS) oder Eventgroups (SOME/IP) ausgetauscht, um die Messungen auf den Discovery-Prozess zu fokussieren und Störeinflüsse durch reguläre Kom-

munikation zu vermeiden. Zur Reduktion der Komplexität haben alle Teilnehmer klare Rollen: Publisheranwendungen sind ausschließlich Publisher und Subscriberanwendungen ausschließlich Subscriber. In realen Systemen können Teilnehmer hingegen sowohl Publisher- als auch Subscriberrollen einnehmen.

5.5.1 Szenario 1: Publisher warten auf Subscriber

Beschreibung Szenario S1 simuliert den Fall, dass ein Publisher bereits existiert und ein Subscriber später hinzukommt. Das ist ein häufiger Praxisfall, z. B. beim Starten neuer Steuergeräte im Fahrzeug.

Beispielhafte Situationen

- Motor läuft bereits, Navigationssystem startet
- Sensoren sind aktiv, Infotainment wird zugeschaltet
- ECU bereits online, Diagnosegerät wird angeschlossen

Ablauf Beim Initialisieren der Testumgebung wird sichergestellt, dass Subscriber erst starten, nachdem alle Publisher gestartet sind. Die Applikationen terminieren, sobald der Discovery-Prozess abgeschlossen ist.

Ziel Untersuchung, wie schnell und zuverlässig ein beitretender Subscriber einen bereits existierenden Publisher entdeckt.

Konfigurationsparameter

CycloneDDS `SPDPInterval`, `HeartbeatInterval`

vSomeIP `cyclic_offer_delay`, `initial_delay_{min, max}`

Tab. 5.7: Übersicht der untersuchten Szenarien

Name	Ablauf	Ziel	Stack	Parameter
Szenario 1: Publisher warten auf Subscriber	Alle Publisher werden sequentiell gestartet, anschließend werden alle Subscriber sequentiell gestartet.	Untersuchung, wie schnell und zuverlässig ein beitretender Subscriber einen bereits existierenden Publisher entdeckt.	CycloneDDS	`LeaseDuration`, `HeartbeatInterval` `SPDPResponseMaxDelay`
			vSomeIP	`cyclic_offer_delay`, `initial_delay_{min,max}`
Szenario 2: Subscriber warten auf Publisher	Alle Subscriber werden sequentiell gestartet, anschließend werden alle Publisher sequentiell gestartet.	Untersuchung, wie schnell und zuverlässig ein bereits beigetretener Subscriber einen dynamisch startenden Publisher entdeckt.	CycloneDDS	`LeaseDuration`, `HeartbeatInterval` `SPDPResponseMaxDelay`
			vSomeIP	`cyclic_offer_delay`, `initial_delay_{min,max}`
Szenario 3: Konkurrierender Systemstart	Publisher und Subscriber werden in zufälliger Reihenfolge gestartet.	Untersuchung, wie schnell und zuverlässig sich verschiedene ECUs bei einem spontanen Fahrzeugstart finden.	CycloneDDS	`LeaseDuration`, `HeartbeatInterval` `SPDPResponseMaxDelay`
			vSomeIP	`cyclic_offer_delay`, `initial_delay_{min,max}`
Szenario 4: Subscriber trennt Verbindung und kommt zurück	Nach dem initialen Discovery-Prozess werden ein oder mehrere Subscriber neugestartet und die Zeit gemessen, bis der Initialzustand wiederhergestellt ist.	Untersuchung, wie schnell und zuverlässig ein Gerät nach einer Unterbrechung eine benötigte ECU wiederentdeckt.	CycloneDDS	`LeaseDuration`, `HeartbeatInterval`, `SPDPResponseMaxDelay`, `DiscoveredLocatorPruneDelay`
			vSomeIP	`cyclic_offer_delay`, `initial_delay_{min,max}`, `ttl`
Szenario 5: Publisher trennt Verbindung und kommt zurück	Nach dem initialen Discovery-Prozess werden ein oder mehrere Publisher neugestartet und die Zeit gemessen, bis der Initialzustand wiederhergestellt ist.	Untersuchung, wie schnell und zuverlässig ein Gerät nach einer Unterbrechung eine benötigte ECU wiederentdeckt.	CycloneDDS	`LeaseDuration`, `HeartbeatInterval`, `SPDPResponseMaxDelay`, `DiscoveredLocatorPruneDelay`
			vSomeIP	`cyclic_offer_delay`, `initial_delay_{min,max}`, `ttl`

5.5.2 Szenario 2: Subscriber warten auf Publisher

Beschreibung Szenario S2 simuliert den Fall, dass ein Subscriber auf das Erscheinen eines neuen Publisher wartet. Das ist relevant, wenn Publisher dynamisch bereitgestellt werden.

Beispielhafte Situationen

- Bordcomputer wartet auf Fahrzeugstatus
- Klimaanlage wartet auf Temperatursensor
- Navigationssystem wartet auf GPS-Signal

Ablauf Beim Initialisieren der Testumgebung wird sichergestellt, dass Publisher erst starten, nachdem alle Subscriber gestartet sind. Die Applikationen terminieren, sobald der Discovery-Prozess abgeschlossen ist.

Ziel Untersuchung, wie schnell und zuverlässig ein bereits beigetretener Subscriber einen dynamisch startenden Publisher entdeckt.

Konfigurationsparameter

CycloneDDS `SPDPInterval`, `HeartbeatInterval`

vSomeIP `cyclic_offer_delay`, `initial_delay_{min, max}`

5.5.3 Szenario 3: Konkurrierender Systemstart

Beschreibung In Szenario S3 starten alle Teilnehmer (Subscriber und Publisher) unabhängig voneinander in undefinierter Reihenfolge. Dieses Szenario ist realitätsnah, da in verteilten (Automotive-)Systemen die Startreihenfolge oft nicht garantiert werden kann.

Beispielhafte Situationen

- Fahrzeugstart – alle Systeme starten gleichzeitig
- Unkoordinierter Bootvorgang nach Stromausfall

Ablauf Beim Initialisieren der Testumgebung starten Publisher- und Subscriber in nicht-deterministischer Reihenfolge (pseudorandomisiert mit einheitlichem Seed). Die Applikationen terminieren, sobald der Discovery-Prozess abgeschlossen ist.

Ziel Untersuchung, wie schnell und zuverlässig sich verschiedene ECUs bei einem spontanen Fahrzeugstart finden.

Konfigurationsparameter

CycloneDDS `SPDPInterval`, `HeartbeatInterval`

vSomeIP `cyclic_offer_delay`, `initial_delay_{min, max}`

5.5.4 Szenario 4: Subscriber trennt Verbindung und kommt zurück

Beschreibung In Szenario 4 verlassen von allen anderen Teilnehmern bereits entdeckte Subscriber das Netzwerk (z. B. durch Neustart oder Netzwerkunterbrechnung) und treten anschließend nach einer Ausfallzeit wieder bei. Hierbei wird getestet, wie schnell und zuverlässig die Rediscovery erfolgt.

Beispielhafte Situationen

- Smartphone verliert Verbindung und verbindet neu
- Navigationssystem startet nach Absturz neu
- Diagnosegerät wird kurz abgesteckt und wieder angeschlossen

Ablauf Beim Initialisieren der Testumgebung wird abgewartet, bis der initiale Discovery-Prozess abgeschlossen ist. Anschließend werden 50% der Subscriber heruntergefahren und nach einer Ausfallzeit wieder hochgefahren. Es wird die Zeit gemessen, wie lange es dauert, bis nach dem Neustart der Initialzustand (alle Teilnehmer haben sich gefunden) wiederhergestellt ist.

Ziel Untersuchung, wie schnell und zuverlässig ein Gerät nach einer Unterbrechung eine benötigte ECU wiederentdeckt und wie sich die Wiederanbindung auf die Discovery-Dauer auswirkt.

Konfigurationsparameter

CycloneDDS `SPDPInterval`, `LeaseDuration`, `DiscoveredLocatorPruneDelay`, `HeartbeatInterval`

vSomeIP `cyclic_offer_delay`, `initial_delay_{min, max}`, `ttl`

5.5.5 Szenario 5: Publisher trennt Verbindung und kommt zurück

Beschreibung In Szenario 5 verlassen bereits entdeckte Publisher das Netzwerk (z. B. durch Neustart oder Netzwerktrennung) und treten anschließend nach einer Ausfallzeit wieder bei. Hierbei wird getestet, wie schnell und zuverlässig die Rediscovery erfolgt.

Beispielhafte Situationen

- Motorsensor fällt aus und startet neu
- Kamerasystem wird kurzzeitig deaktiviert

Ablauf Beim Initialisieren der Testumgebung wird abgewartet, bis der initiale Discovery-Prozess abgeschlossen ist. Anschließend werden 50% der Publisher heruntergefahren und nach einer Ausfallzeit wieder hochgefahren. Es wird die Zeit gemessen, wie lange es dauert, bis nach dem Neustart der Initialzustand (alle Teilnehmer haben sich gefunden) wiederhergestellt ist.

Ziel Untersuchung, wie schnell und zuverlässig ein bereits beigetretenes Gerät eine nach Unterbrechung wieder erscheinende ECU wiederentdeckt und wie sich die Wiederanbindung auf die Discovery-Dauer auswirkt.

Konfigurationsparameter

CycloneDDS `SPDPInterval`, `LeaseDuration`, `DiscoveredLocatorPruneDelay`, `HeartbeatInterval`

vSomeIP `cyclic_offer_delay`, `initial_delay_{min, max}`, `ttl`

6 Ergebnisse und Evaluation

Im folgenden Kapitel werden die Ergebnisse der durchgeführten Testläufe präsentiert und analysiert. Zuerst werden die Auswertungsmethoden erläutert. Anschließend werden jeweils für beide Middlewares zusammenfassend das Abschneiden in den Szenarien evaluiert und die Auswirkungen der verschiedenen Parameterkombinationen diskutiert. Abschließend wird ein direkter Vergleich zwischen beiden Middlewares gezogen, die Kernerkenntnisse zusammengefasst sowie Limitationen der Messstudie besprochen. Aus Gründen der Übersichtlichkeit werden in diesem Kapitel nur die relevantesten Ergebnisse dargestellt. Detaillierte Ergebnisse aller Testläufe sind im Anhang A zu finden.

6.1 Auswertung der Metriken

Zur statistischen Auswertung wurden in jedem Testlauf automatisiert Metriken erfasst und darauf basierend folgende Statistiken generiert:

- Mittelwert
- Minimum, Maximum
- P25-, P50-, P75-, P90-, P99-, P99.9-Perzentil
- Standardabweichung, Variationskoeffizient

Auf Publisherseite beziehen sich alle Werte immer auf die Discovery *all* seiner Subscriber. So gibt bspw. der Mittelwert an, wie schnell ein Publisher im Mittel alle seine Subscriber innerhalb eines Testlaufs gefunden hat. Auf Subscriberseite hingegen gibt der Mittelwert an, wie schnell ein Subscriber im Mittel seinen Publisher gefunden hat. Analog verhält es sich mit den Perzentilen: das P99-Perzentil gibt an, wie lange es gedauert hat, bis 99 % aller Publisher alle ihre Subscriber gefunden haben. Beim Subscriber gibt es den Zeitraum an, in dem 99 % der Subscriber ihren zugewiesenen Publisher entdeckt haben.

Analog verhält es sich bei den anderen Perzentilen. Die Perzentile werden mithilfe von Boxplots dargestellt. Die Enden der Whiskers der Boxplots stellen in der hier verwendeten Darstellung die Minimal- und Maximalwerte dar. Die Standardabweichung macht eine Aussage darüber, inwiefern die Messergebnisse absolut gestreut sind. Der Variationskoeffizient ist der Quotient aus Standardabweichung und Mittelwert und bildet damit ein relatives Streuungsmaß ohne Maßeinheit. Ein kleiner Wert deutet darauf hin, dass die einzelnen Discovery-Zeiten nah beieinander liegen, während ein großer Wert auf eine hohe Variabilität hinweist.

In den Abbildungen 6.2 und 6.3 werden auf Basis der ausgewerteten Metriken die besten Testdurchläufe je Szenario und Konstellation dargestellt. Dazu wurde für jeden Testlauf ein gewichteter Score generiert, der mit einem Faktor von 50 % die Gesamtlatenz und mit je je 25 % den Median der Publisher- und Subscriberlatenzverteilung addiert. Je niedriger der Score, desto besser wird der Durchlauf bewertet. Auf diesem Score basieren auch die besten Testdurchläufe der Tabellen im Anhang A.

6.2 Auswertung der Parametermessungen

Tabelle 6.1 listet die Anzahl unterschiedlicher getesteter Parameterkombinationen nach Szenario und Middleware auf. Da ein Teil der getesteten Parameter nur in Kombination mit dem Neustart und der Wiederentdeckung von Teilnehmern sinnvoll zu testen ist, weisen die Szenarien S4 und S5 mehr Parameterkombinationen auf. Parameterkombinationen, welche keinen oder nur wenig Einfluss (ausgehend von der Standardeinstellung) haben, werden zu besseren Übersichtlichkeit nicht in die tiefergehende Analyse übernommen.

Tab. 6.1: Anzahl Parameterkombinationen nach Middleware und Szenarien

Szenarien	**vSomeIP**	**CycloneDDS**
S1-3 (Szenarien ohne Neustart)	12	12
S4-5 (Szenarien mit Neustart)	27	35

Da der Einfluss der Parameter auf die Latenz nicht immer ohne Weiteres visuell durch Heatmaps oder andere Diagramme zu ermitteln ist, wurde zur Unterstützung und Quantifizierung eine lineare Regression mittels *Methode der kleinsten Quadrate* (MKQ) durch-

geführt. Für jede Kombination aus Szenario (S1–S5) und Konstellation (K1–K5) wurde mit der Python-Bibliothek `statsmodels` ein separates Regressionsmodell trainiert, wobei je nach Middleware die spezifischen Konfigurationsparameter als unabhängige Variablen und die Gesamtlatenz Δt_{total} als Zielvariable dienen. Nicht gesetzte Parameter wurden für die Analyse mit den jeweiligen Voreinstellungen (siehe Tabellen 5.3 und 5.4) aufgefüllt. Die Regressionsanalyse berechnet unter anderem einen Koeffizienten für jeden Parameter, welcher die Richtung und Stärke des Einflusses auf die Zielvariable angibt sowie einen p-Wert, der die statistische Signifikanz des Parameters angibt. Die Ergebnisse der Regressionsanalyse werden in Form einer fünfstufig farbcodierten Matrix dargestellt. Dabei orientieren sich die Werte einerseits am Koeffizienten sowie am p-Wert. Die genauen Werte für die Farbcodierung finden sich in der Legende der jeweiligen Matrix.

6.3 CycloneDDS: Ergebnisse

Im Folgenden werden die Ergebnisse der Testläufe mit CycloneDDS analysiert. Zunächst werden die Szenarien und Konstellationen analysiert, bevor anschließend die Auswirkungen der verschiedenen Konfigurationsparameter diskutiert werden. Besonderes Augenmerk liegt dabei auf der Skalierbarkeit und den beobachteten Effekten bei steigender Teilnehmerzahl sowie auf den spezifischen Herausforderungen, die sich im Testumfeld gezeigt haben.

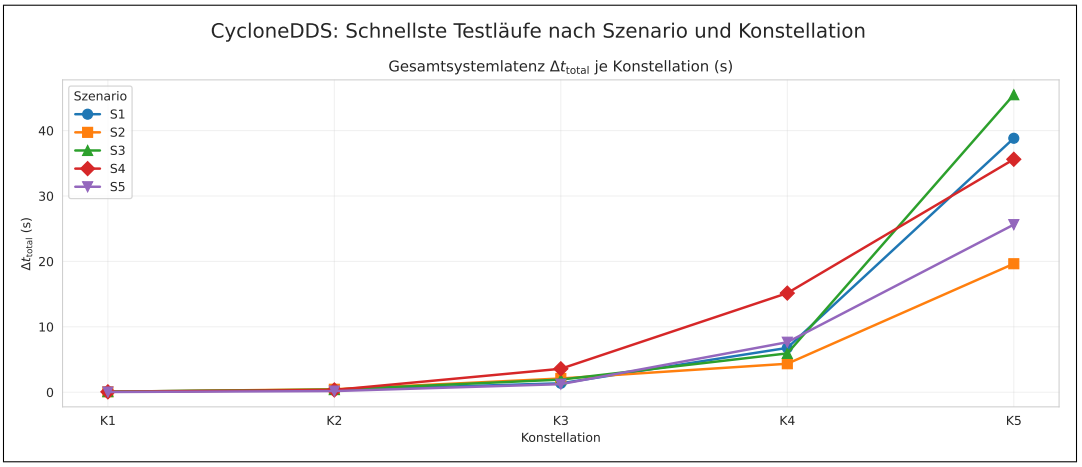
6.3.1 Szenarioanalyse

Abb. 6.1 zeigt die Gesamtlatenz[1] des am besten bewerteten Testlaufs je Szenario für alle Konstellationen in CycloneDDS. Während es in kleineren Konstellationen (K1–K3) moderate Unterschiede zwischen den Szenarien gibt, zeigen sich ab K4 massive Unterschiede und Latenzen im Bereich von 5 bis fast 45 Sekunden. Betrachtet man die Verteilung der einzelnen Latenzen auf Publisher- und Subscriberseite szenarioübergreifend, so zeigt sich, dass es ab Konstellation K4 zu starken Ausreißern mit sehr hohen Latenzwerten kommt, welche sich in der Gesamtlatenz niederschlagen. Dies ist auf das Leistungslimit der Testumgebung zurückzuführen: 400 (K4) bzw. 600 (K5) CycloneDDS-Applikationen lasten alle CPU-Kerne des Systems zu 100 % aus, weshalb Nachrichten verzögert verarbeitet

[1] Zur besseren Lesbarkeit werden statt der Formeln in den folgenden Abschnitten die Begriffe „Gesamtlatenz“ für Δt_{total}, „Publisherlatenzen“ für alle $\Delta t_{\text{SD}}^{(p)} \mid p \in \mathcal{P}$ und „Subscriberlatenzen“ für alle $\Delta t_{\text{SD}}^{(s)} \mid s \in \mathcal{S}$ verwendet.

und/oder wegen Pufferüberläufen mehrfach gesendet werden. In Abschnitt 6.6 wird detaillierter auf diesen Aspekt eingegangen.

Abb. 6.1: CycloneDDS: Schnellste Testläufe (Systemlatenz) pro Szenario über alle Teilnehmerkonstellationen



Da die Konstellationen K4 und K5 nicht für eine zielführende Evaluation geeignet sind, werden für die Szenarien S1–S5 nur die Konstellationen K1–K3 analysiert. Abb. 6.2 zeigt die Gesamtlatenzen der schnellsten Testlaufs je Szenario sowie die jeweils zugehörige Verteilung der Publisher- und Subscriberlatenzen.

In der kleinsten Konstellation K1 sind bei der Gesamtlatenz Unterschiede im Millisekundenbereich zwischen den Szenarien festzustellen. Diese bewegen sich alle im Bereich von unter 100 ms, mit S5 als schnellster (35 ms) und S1 (94 ms) als langsamster Konstellation.

In Konstellation K2 ist wieder S5 am schnellsten (177 ms), wohingegen S2 mit 457 ms am schlechtesten abschneidet. In S2 steigt bei den Subscriberlatenzen die Streuung bereits deutlich: mit einem Variationskoeffizient von 1,15 liegt die Standardabweichung bei 115 % des Mittelwerts.

In Konstellation K3 ist wieder S5 am schnellsten (1,23 s), wohingegen S4 mit 3,64 s deutlich schlechter abschneidet und somit fast dreimal so langsam wie S5 ist. Dieser starke Unterschied hängt sehr wahrscheinlich mit der unterschiedlichen Anzahl betroffener Teilnehmer zusammen: In S4 werden 50 % der Subscriber neu gestartet, was bei der gewählten Topologie (4 Subscriber je Publisher in K3) 100 von 200 Subscribern entspricht. In S5 hingegen werden 50 % der Publisher neu gestartet, also lediglich 25 von 50

Publishern. Die höhere absolute Anzahl neu startender Subscriber führt in S4 zu einer deutlich längeren Gesamtlatenz. In beiden Szenarien müssen 100 SPDP/SEDP-Prozesse neu durchlaufen werden, jedoch ist S4 mit der größeren Anzahl neu startender Anwendungen fast 3 mal so langsam. Hier zeigt sich erneut die Limitation der Testumgebung bzw. der Ressourcenverbrauch der getesteten DDS-Implementierung.

In diesem Zusammenhang ist auch das Verhältnis aus übertragener Datenmenge und Gesamtlatenzen interessant:

- **K1:** ca. 1,4 MiB, 35-94 ms
- **K2:** ca. 5,5 MiB, 177-457 ms
- **K3:** ca. 26 MiB, 1,2-3,6 s

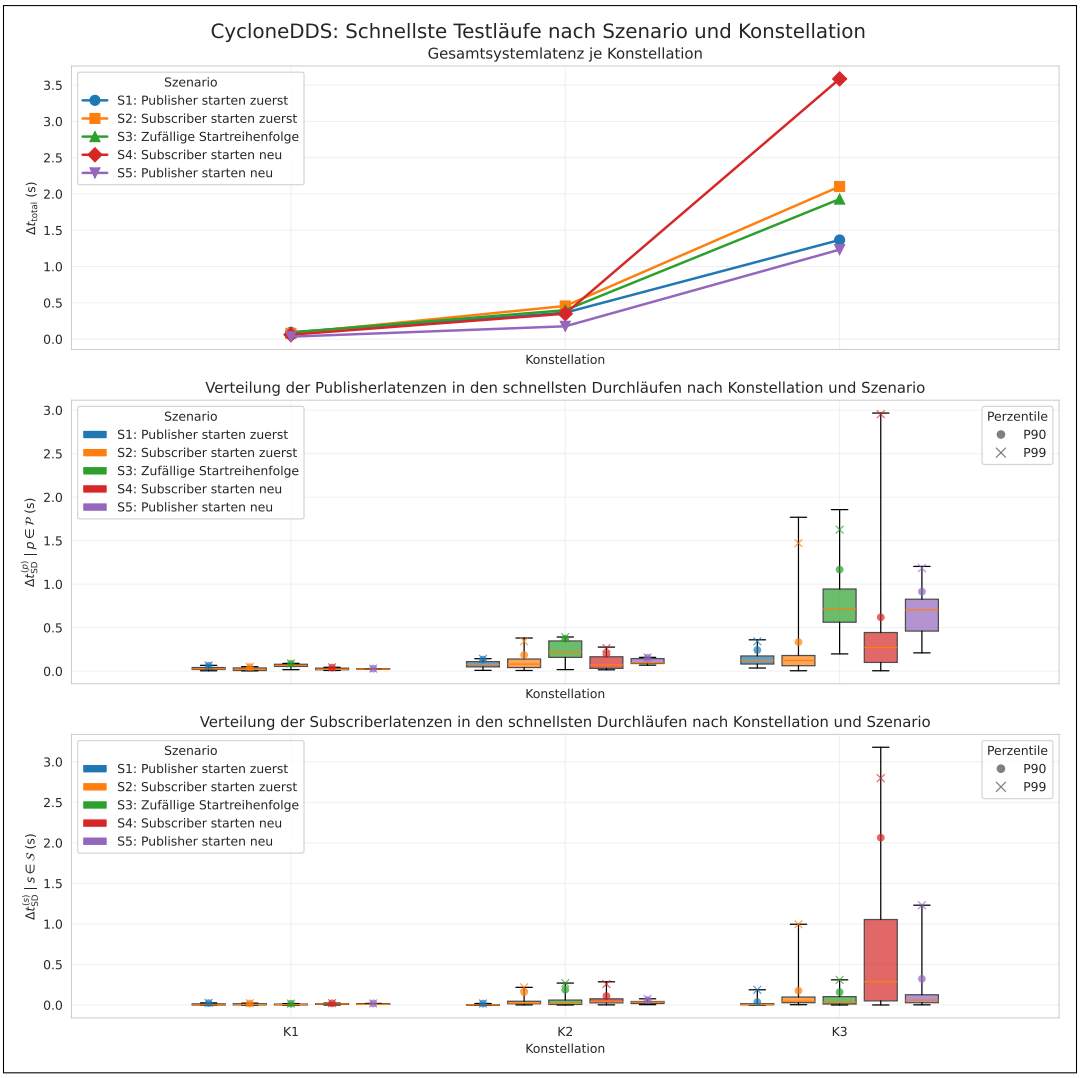
Die Zunahme der übertragenen Bytes (Faktor 19x von K1 zu K3) korreliert nicht mit der Latenzsteigerung (Faktor 38x von K1 zu K3). Dies lässt darauf schließen, dass der Flaschenhals nicht im Netzwerkaufbau liegt, da 26 MiB über einen Zeitraum von mehreren Sekunden für einen virtuellen Kernelswitch keine anspruchsvolle Aufgabe sind: Die Mininet-Dokumentation gibt eine Bandbreite von ca. 2 Gb/s auf „bescheidener“ Hardware an [16].

Darüber hinaus zeigen sich in K3 massive *Tail Latency*-Probleme: einige wenige Teilnehmer weisen extrem hohe Discoverylatenzen auf, wodurch die Gesamtlatenz steigt. So liegt beispielsweise der Median für Subscriberlatenzen in S2 bei 46 ms, der Maximalwert hingegen bei 996 ms. Dies schlägt sich auch im Variationskoeffizient nieder, der in K3 für Subscriber zwischen 122 % und 181 % liegt.

Publisher zeigen im Gegensatz zu Subscribern weniger Streuung in den Discoverylatenzen. Hier liegt die Vermutung nahe, dass dies mit den Konstellationen selbst zu tun hat: Publisher warten auf Subscriber, die sich ausschließlich für die Topic dieses Publishers interessieren. Subscriber hingegen warten auf Publisher, die 2 bis 5 (je nach Konstellation) Subscriber bedienen müssen. Dementsprechend hat bei Subscribern eine verspätete Nachricht bei der Zeitmessung wesentlich mehr Gewicht als bei Publishern, bei denen die Publisherlatenz aus mehr Zeitstempeln gebildet wird. Diese statistische Mittelung ist daher wahrscheinlich auf eine Eigenheit des Messaufbaus zurückzuführen.

In den Szenarien S2, S4 und S5 lässt sich in K3 für Subscriber außerdem ein „First-Come-First-Served“-Effekt beobachten. Während die unteren Quartile meist sehr schnell bedient werden, sind die oberen Perzentile hingegen sehr langsam. Dies lässt darauf

Abb. 6.2: CycloneDDS: Schnellste Testläufe (Systemlatenz) pro Szenario über Teilnehmerkonstellationen K1–K3



schließen, dass Nachrichtenpuffer in den CycloneDDS-Anwendungen über die Zeit ausgelastet werden. Die Vermutung liegt nahe, dass die ersten bei den Publishern eintreffenden SPDP-Nachrichten noch umgehend mit einer SEDP-Nachricht beantwortet werden, bis sich über die Zeit die Puffer füllen und Nachrichten auf Publisherseite verworfen werden. Erst nach einem HEARTBEAT-ACKNACK-Nachrichtenaustausch zwischen beiden Teilnehmern kann die SEDP-Nachricht dann erneut versendet werden (*Retransmission*).

6.3.2 Parameterstudie

In Anhang A sind Heatmaps zu finden, welche für jede Parameterkombination die über alle Iterationen gemittelte Gesamtlatenz je Szenario und Konstellation darstellen. Je nach Parameter sind die Effekte mehr oder weniger gut visuell zu erkennen. Daher sind zur besseren Visualisierung in Tabelle 6.2 die Ergebnisse der linearen Regression zur Analyse der Parameterwirkungen in CycloneDDs aufgeführt.

Hierbei ist auffällig, dass eine Erhöhung von `HeartbeatInterval` und `SPDPResponseMaxDelay` in den kleinen Konstellationen K1–K2 die Gesamtlatenz erhöht. Bei `HeartbeatInterval` ist dies interessant, da Heartbeat-Nachrichten in DDS primär zur Erkennung von verlorenen Nachrichten und ausgefallenen Teilnehmern dienen und daher in Mininet ohne Paketverluste keinen größeren Einfluss auf die Discovery-Zeiten haben sollten. Die Erwartung wäre gewesen, dass ein höheres `HeartbeatInterval` die Nachrichtenlast reduziert und somit die Discovery-Zeiten verbessert oder in den kleineren Konstellationen zumindest neutral wirkt.

Bei `SPDPResponseMaxDelay` hingegen ist die Latenzerhöhung plausibel, da eine Erhöhung dieses Wertes die Zeitspanne verlängert, in der auf SPDP-Nachrichten geantwortet wird. Da die Anzahl Netzwerkteilnehmer in diesen Konstellationen gering ist, überwiegt der negative Effekt der verlängerten Wartezeit auf SPDP-Antworten gegenüber dem positiven Effekt einer verringerten Nachrichtenlast. In den Konstellationen K3–K5 hingegen ist kein signifikanter Effekt durch `SPDPResponseMaxDelay` mehr festzustellen. Ein möglicher Grund hierfür ist, dass in diesen Konstellationen die Nachrichtenlast durch die hohe Anzahl an Teilnehmern (und damit HEARTBEAT-, ACKNACK- und SEDP-Nachrichten) ohnehin schon so hoch ist, dass eine weitere Reduktion der SPDP-Nachrichtenlast durch eine Erhöhung von `SPDPResponseMaxDelay` keine nennenswerte Verbesserung mehr bringt.

Insgesamt ist zu erkennen, dass es eine Art „Umschwung“ bezüglich der Parameterwirkungen im Bereich der Konstellation K3 gibt. Dies steht höchstwahrscheinlich in Zusammenhang mit den im vorherigen Abschnitt beschriebenen Leistungseinbrüchen bei steigender Teilnehmerzahl. Da `HeartbeatInterval` und `SPDPResponseMaxDelay` die Nachrichtenlast reduzieren, wandelt sich ihr Effekt im Bereich von K3. In den Neustart-Szenarien S4 und S5 ist führen sie zu einer starken Reduktion der Gesamtlatenz, da bei Wiederkehr die bestehenden Teilnehmer wesentlich weniger Nachrichten untereinander austauschen und die Nachrichtenpuffer somit weniger stark ausgelastet sind.

Der Parameter `LeaseDuration` zeigt in den Konstellationen K1–K3 keine signifikanten Effekte. Dies ist insofern interessant, da das `SPDPInterval` mit ca. 80 % implizit durch den Parameter `LeaseDuration` gesetzt wird. Der Parameter wird erst in den Konstellationen K4–K5 negativ signifikant; in K4 in den Szenarien S4–S5 sowie in K5 in allen Szenarien. Dies ist ein erwartbarer Effekt: kehrt z. B. ein Subscriber nach Neustart zurück und hat gerade eine für ihn relevante SPDP-Nachricht von seinem Publisher verpasst, so muss er bis zu 80 % der LeaseDuration warten, bis er erneut Teilnehmerinformationen erhält. I. d. R. wird der Publisher zwar auf eine SPDP-Nachricht des zurückgekehrten Subscribers mit einer SPDP-Nachricht außerhalb des periodischen Intervalls antworten, allerdings kann dies bei hoher Systemauslastung, die in den Konstellationen K4–K5 vorliegt, stark verzögert sein.

Tab. 6.2: Parameterwirkungen in CycloneDDS (Signifikanz u. Einflussrichtung bzgl. Systemlatenz)

Parameter		K1	K2	K3	K4	K5
HeartbeatInterval	**S1**	↑↑	↑↑	o	o	o
	S2	↑↑	↑↑	↑↑	o	o
	S3	↑↑	↑↑	o	o	o
	S4	↑↑	↑↑	↓↓	↓↓	o
	S5	↑↑	↑↑	↓↓	↓↓	↓↓
SPDPResponseMaxDelay	**S1**	↑	↑↑	↑↑	o	o
	S2	↑↑	↑↑	o	↑	o
	S3	↑↑	↑↑	o	o	o
	S4	↑↑	↑↑	o	o	o
	S5	↑↑	↑↑	↑↑	o	o
LeaseDuration	**S1**	o	o	o	o	↑↑
	S2	o	o	o	o	↑↑
	S3	o	o	o	o	↑↑
	S4	o	o	o	↑↑	↑↑
	S5	o	o	o	↑↑	↑↑
DiscoveredLocatorPruneDelay	**S4**	o	o	o	o	o
	S5	o	o	o	o	o
Downtime-exceed	**S4**	o	o	↓	o	o
	S5	o	o	o	↓	o

Legende:
↑↑ stark positiv ($p < 0.01$), ↑ positiv ($p < 0.05$), o nicht signifikant, ↓ negativ ($p < 0.05$), ↓↓ stark negativ ($p < 0.01$).
Positive Werte bedeuten: höherer Parameterwert → höhere Latenz.
Negative Werte bedeuten: höherer Parameterwert → niedrigere Latenz.

Der Parameter `DiscoveredLocatorPruneDelay` zeigt in keinem Testfall einen signifikanten Effekt. Die Annahme war hierbei, dass es einen zeitlichen Effekt auf die Rediscovery haben könnte, wenn das `DiscoveredLocatorPruneDelay` statt der voreingestellten 60 s nur 0 s betragen würde, weil Locator-Informationen direkt nach ablauf der LeaseDuration direkt gelöscht würden.

Der Parameter `Downtime-exceed` stellt keinen CycloneDDS-Parameter, sondern ein Parameter für die Testumgebung dar. Er beschreibt, ob die Ausfallzeit in S4 und S5 die LeaseDuration überschritten hat und zeigt nur sehr schwache Effekte in S4 und S5. Hier liegt die Vermutung nahe, dass es sich um Messartefakte handelt, da auch die visuelle Analyse der Heatmaps in Anhang A keinen klaren Effekt erkennen lässt.

6.4 vSomeIP: Ergebnisse

Im Folgenden werden die Ergebnisse der vSomeIP-Testläufe analysiert. Analog zur vorherigen Auswertung werden zunächst die Szenarien zusammenfassend betrachtet, bevor anschließend die Auswirkungen der verschiedenen Konfigurationsparameter diskutiert werden. Im Gegensatz zu CycloneDDS zeigen sich bei vSomeIP deutlich besser interpretierbarere Zusammenhänge zwischen Parameterkonfiguration und Discovery-Latenz.

6.4.1 Szenarien

Abb. 6.3 zeigt die schnellsten Testdurchläufe bzgl. der Gesamtlatenz sowie die Verteilung der zugehörigen Publisher- und Subscriberlatenzen für diese Durchläufe.

In den kleineren Konstellationen (K1–K3) bewegen sich die Discoverylatenzen auf sehr niedrigem Niveau. So liegt die Gesamtlatenz in K1 im Bereich von wenigen Millisekunden, mit minimalen Unterschieden zwischen den Szenarien. Auch in K2 und K3 steigen die Latenzen nur moderat an und bleiben im Bereich von wenigen bis einigen Zehntelsekunden.

Mit zunehmender Teilnehmerzahl (K4, K5) steigen die Discovery-Latenzen erwartungsgemäß an. In K5 werden Latenzen im Bereich von mehreren hundert Millisekunden bis über eine Sekunde erreicht. In diesen Szenarien steigt auch die Verteilung der Publisher- und Subscriberlatenzen. In allen Fällen ist die Spanne in S3 am höchsten (der Variationskoeffizient für Publisherlatenzen liegt in K5 bei 284 %), was einerseits damit zusammenhängt,

dass durch die randomisierte Startreihenfolge manche Subscriber wesentlich früher oder später als ihr zugewiesener Publisher starten. Andererseits hängt dies mit den Parametern `initial_delay_min` und `initial_delay_max` zusammen, welche eine randomisierte Startverzögerung innerhalb des spezfizierten Intervalls verursachen. In diesem Kontext ergibt es bspw. für Szenario S3 in Konstellation K5 absolut Sinn, dass die maximale Publisherlatenz knapp unter 2000 ms liegt, da der Parameter `initial_delay_max` dort im schnellsten Durchlauf auf 2000 ms gesetzt wurde und einer der zugewiesenen Subscriber die 2000 ms „ausgereizt“ hat. Die genannten Parameter werden im folgenden Abschnitt 6.4.2 nochmals besprochen.

In den Szenarien S1–S3 sind generell hohe Tail Latencies zu beobachten: so beträgt der Median der Publisher- und Subscriberlatenzen oftmals ein Fünftel oder weniger des Maximalwertes. Dies könnte damit zusammenhängen, dass, durch die Startreihenfolge bedingt, manche einander zugewiesene Teilnehmer sich während der Repetition Phase verpassen und demnach ihre Discovery verzögert wird.

Insgesamt auffällig ist, dass die Gesamtlatenz in vSomeIP nahezu linear mit der Teilnehmerzahl skaliert. Eine Verdopplung der Teilnehmerzahl führt in etwa zu einer Verdopplung der Discovery-Zeit. Im Gegensatz zu CycloneDDS treten bei vSomeIP keine extremen Ausreißer oder exponentiellen Latenzsteigerungen auf. Die Systemlatenz bleibt auch in großen Konstellationen im Bereich von etwa 1-2 Sekunden. Dies zeigt sich auch bei den Neustart-Szenarios S4 und S5: Diese sind durchgängig am schnellsten, weil sie schlichtweg im Verhältnis weniger Teilnehmer umfassen. Der Aspekt des Neustarts macht hierbei keinen Unterschied zu den anderen Szenarien.

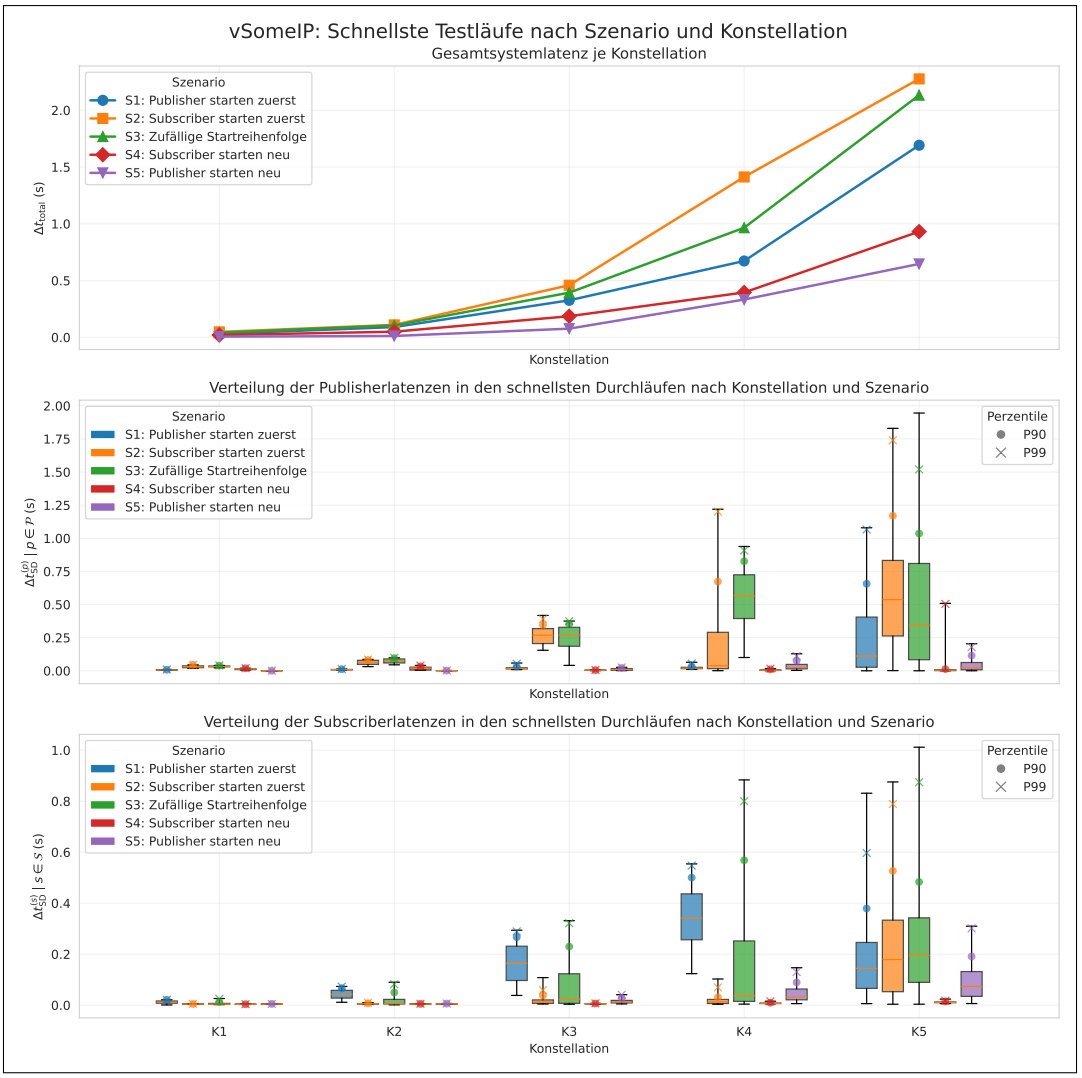
Insgesamt zeigt sich, dass vSomeIP/SOME/IP Service Discovery in der getesteten Umgebung sehr effizient arbeitet und auch bei steigender Teilnehmerzahl szenarienübergreifend eine gute Skalierbarkeit aufweist.

6.4.2 Parameterstudie

In den Testläufen zu vSomeIP sind die Effekte der Parameter visuell besser zu erkennen als bei CycloneDDS. Es zeigt sich klar, dass `initial_delay_min` und `initial_delay_max` starken Effekt auf die Service Discovery haben.

Die lineare Regressionsanalyse für die Parameterkombinationen ist im Fall von vSomeIP eingeschränkt zu interpretieren. Dadurch, dass die Parameter `initial_delay_min` und

Abb. 6.3: vSomeIP: Schnellste Testläufe (Systemlatenz) pro Szenario und Teilnehmerkonstellation



`initial_delay_max` immer gemeinsam gesetzt wurden, sind sie keine unabhängigen Variablen, was in einer starken Multikollinearität für die beiden Parameter resultiert. Daher wurden beide Parameter aus der Matrix 6.3 entfernt. Ihr Effekt ist jedoch visuell in den Heatmaps und Tabellen in Anhang A gut zu erkennen.

Die Kombination eines `initial_delay_min` von 0 ms und eines `initial_delay_max` von 0 ms (also ein unmittelbarer Start der Discovery) führt in den drei kleineren Konstellationen (K1 bis K3) zu den besten Ergebnissen, während in der größten Konstellation (K4)

die Kombination eines `initial_delay_min` von 500 ms und eines `initial_delay_max` von 2000 ms die besten Ergebnisse erzielt. Dies illustriert die Relevanz und das Optimierungspotential der Parameter `initial_delay_min` und `initial_delay_max` in vSomeIP: In kleineren Topologien kann direkt mit der Discovery begonnen werden, während durch eine Streuung der Startzeiten in größeren Topologien eine Nachrichtenexplosion im Netzwerk und eine Überlastung der Endpunkte vermieden werden kann.

Der Parameter `cyclic_offer_delay` wirkt sich je nach Anwendungsfall sehr unterschiedlich aus. So wirkt sich ein hohes `cyclic_offer_delay` in S4 immer eine stark negative Wirkung. Dies hängt damit zusammen, dass in S4 Subscriber neustarten und von OfferService-Nachrichten der Publisher abhängig sind. Ein höheres Delay sorgt hier für Verzögerungen, was auch der Erwartungshaltung entspricht. In S1 ist es analog: auch sind Publisher bereits gestartet und befinden sich womöglich schon in der Main Phase, wenn die Subscriber starten. Seltenere OfferService-Nachrichten haben daher auch hier einen negativen Effekt. In K5 ist der Effekt umgedreht: Hier zeigt sich, dass ein höheres `cyclic_offer_delay` die Nachrichtenlast reduzieren kann, was sich positiv auf die Gesamtlatenz auswirkt.

Tab. 6.3: Parameterwirkungen in vSomeIP (Signifikanz u. Einflussrichtung bzgl. Systemlatenz)

Parameter		K1	K2	K3	K4	K5
cyclic_offer_delay	**S1**	↑↑	↑	↑	↑	↓
	S2	↑↑	↑	o	o	↓↓
	S3	↑↑	↑↑	↑↑	o	↓↓
	S4	↑↑	↑↑	↑↑	↑↑	↑↑
	S5	↓	o	o	o	↓↓
ttl	**S4**	o	o	o	o	o
	S5	o	o	o	o	o
Downtime-exceed	**S4**	o	o	o	o	↓
	S5	o	o	o	o	o

Legende:
↑↑ stark positiv ($p < 0.01$), ↑ positiv ($p < 0.05$), o nicht signifikant, ↓ negativ ($p < 0.05$), ↓↓ stark negativ ($p < 0.01$).
Positive Werte bedeuten: höherer Parameterwert → höhere Latenz.
Negative Werte bedeuten: höherer Parameterwert → niedrigere Latenz.

Die Parameter `ttl` und `downtime_exceed` haben nahezu keinen signifikanten Effekt. Bei `ttl` war dies zu erwarten, da dieser Parameter nur gesetzt wurde, um den Standardwert 0xFFFFFF (TTL bis zum nächsten Start) zu überschreiben und die Ausfallzeit die TTL

überschreiten zu lassen. Bei `downtime_exceed` hingegen war die Erwartung, dass bei Überschreiten der TTL die Rediscovery langsamer erfolgt, was sich jedoch nicht bestätigt hat.

Die Untersuchung der Parameter zeigt insgesamt, dass sich `cyclic_offer_delay` , `initial_delay_min` und `initial_delay_max` sehr gut zur Steuerung des Verhaltens der Service Discovery eignen und es sich lohnt, empirisch zu ermitteln, welche Werte für den ausgewählten Anwendungszweck sinnvoll sind. Im Automotive-Kontext ist sehr relevant, wie schnell sich nach Fahrzeugstart parallel hochfahrende ECUs in einem bandbreitenlimitierten Fahrzeugnetzwerk ohne Netzwerküberlastung finden können und abgestürzte Teilnehmer nach Neustart wiederverbinden können. Über die genannten Parameter lässt sich dies sehr gut steuern.

6.5 Übergreifende Analyse

Der folgende Vergleich orientiert sich an den in Kapitel 4 aufgeführten, nicht-funktionalen Anforderungen an Middlewares im Automobilkontext.

6.5.1 Performanz und Determinismus

vSomeIP schneidet in allen Szenarien und bezüglich der Gesamtlatenz besser ab als CycloneDDS. Dies ist insbesondere in den kleinen Konstellationen K1 und K2 zu beachten, in denen die Testumgebung bei CycloneDDS noch nicht ans Leistungslimit gerät und demnach die Vergleichsmetriken noch nicht verfälscht.

Dies lässt sich mit hoher Wahrscheinlichkeit mit der geringeren Nachrichtengröße und -komplexität erklären, wodurch die Anwendungen weniger Rechenaufwand bzgl. der Verarbeitung haben. Des Weiteren ist das Nachrichtenaufkommen bei CycloneDDS um ein Vielfaches höher.

Bei beiden Middlewares lässt sich für die schnellsten Durchläufe in größeren Konstellationen eine höhere Streuung bzgl. der einzelnen Publisher- und Subscriberlatenzen beobachten. Bei CycloneDDS hängt dies insbesondere mit der sehr hohen Systemlast und den damit verursachten Leistungseinbrüchen zusammen, während es bei vSomeIP auf die verwendeten Werte von `initial_delay_{min, max}` zurückzuführen ist. Dies deutet

darauf hin, dass diese Parameter auch in CycloneDDS einen positiven Effekt auf das Discovery-Verhalten haben könnten.

In diesem Kontext ist nochmals die Studie von *Klüner et al.* [14] zum Vergleich heranzuziehen: in der dort durchgeführten Messstudie schneidet vSomeIP im Vergleich zu FastDDS und Zenoh bzgl. der Discovery-Geschwindigkeit durchgehend am schlechtesten ab, was sich in der vorliegenden Arbeit keinster Weise bestätigt. Der Verdacht liegt nahe, dass in der dortigen Studie die Einstellung von `initial_delay_{min, max}` nicht optimiert wurde, und diese für die kleine Anzahl dort getesteter Knoten unvorteilhaft ist. Generell ist zu erwähnen, dass die Standardvoreinstellung von `initial_delay_{min, max}` in vSomeIP 0 ms und 3000 ms beträgt, was ein sehr großzügiges Intervall ist.

6.5.2 Ressourceneffizienz und Skalierbarkeit

Die Ergebnisse aller getesteten Szenarien zeigen, dass CycloneDDS wesentlich ressourcenintensiver arbeitet als vSomeIP. Dies zeigt sich insbesondere in der Systemlatenz, welche in Abbildung 6.4 beispielhaft für Szenario S3 dargestellt ist. Während in kleineren Konstellationen die Latenzen marginal sind, steigen diese in CycloneDDS mit zunehmender Teilnehmeranzahl stark an. vSomeIP hingegen skaliert nahezu linear mit steigender Teilnehmerzahl.

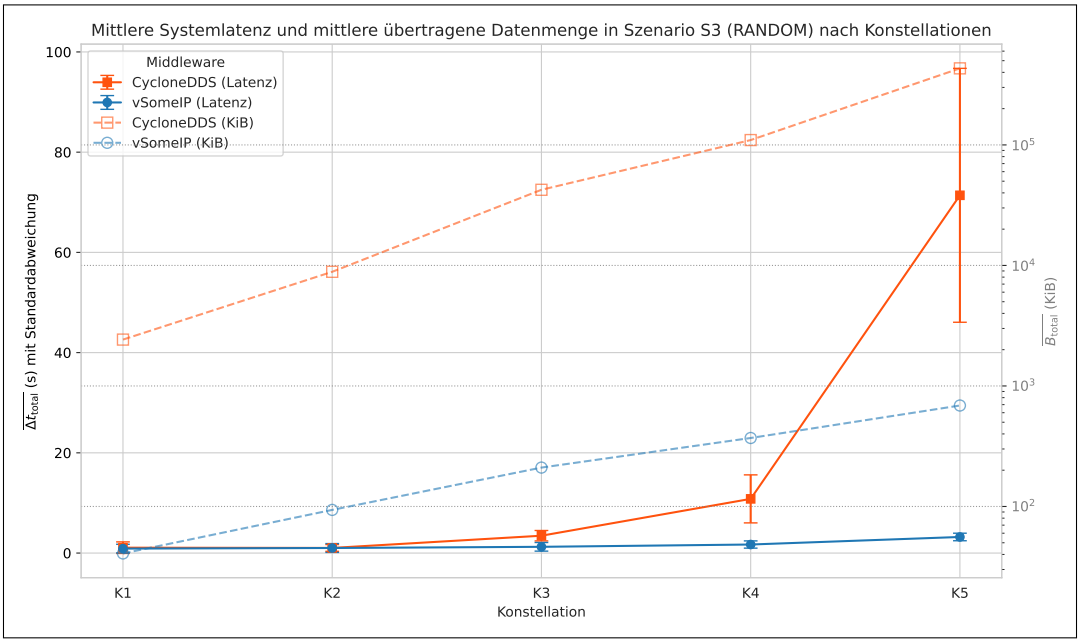
Weitergehend zeigt Abb. 6.4, dass die Konfigurationsparameter in CycloneDDS vor allem bei größeren Konstellationen massiven Einfluss auf die Discoverylatenz haben. Die Latenz schwankt in der größten Konstellation je nach Konfiguration um mehr als 20 s um den Mittelwert. Insgesamt ist die Skalierbarkeit für CycloneDDS jedoch nicht abschließend zu bewerten, da das Leistungslimit der Testumgebung in den höheren Konstellationen die Messergebnisse massiv verfälscht.

Weitergehend lassen sich große Unterschiede bei der übertragenen Datenmenge beobachten. Während vSomeIP in Konstellation K5 im schnellsten Testlauf etwa 655 KiB an Daten überträgt, sind es bei CycloneDDS mehr als 120 MiB (siehe Tabelle A.2), wobei hier nur SPDP- und SEDP-Nachrichten berücksichtigt werden. Mit Heartbeat- und Acknack-Nachrichten wäre die Datenmenge wesentlich höher.

Hierbei ist zu beachten, dass das explosionsartige Wachstum der Latenz unter anderem durch das Leistungslimit der Testumgebung verursacht wird. Während der Messungen war für CycloneDDS ab Konstellation K3 während der Discovery-Phase eine

CPU-Auslastung von nahezu 100 % zu beobachten. Dies hängt zum einen mit der hohen Anzahl an Nachrichten zusammen, die CycloneDDS im Vergleich zu vSomeIP generiert. Zum anderen ist die Implementierung von CycloneDDS offenbar nicht optimal auf ressourcenbeschränkte Umgebungen ausgelegt.

Abb. 6.4: Mittlere Systemlatenz und übertragene Datenmenge in Szenario S3 je Teilnehmerkonstellation



6.5.3 Robustheit und Zuverlässigkeit

Die Szenarien S4 und S5 sollen die Robustheit und Zuverlässigkeit bzgl. der Erkennung von Ausfällen mit anschließender Rediscovery untersuchen. Hierbei zeigt sich, dass beide Middlewares grundsätzlich in der Lage sind, ausgefallene Teilnehmer zu erkennen und nach deren Rückkehr erneut zu entdecken.

In vSomeIP erfolgt die Rediscovery nach Neustart eines Teilnehmers nahezu identisch zur initialen Discovery. Die Gesamtlatenzen in S4 und S5 sind durchgängig niedriger als in den Szenarien S1–S3, was primär auf die geringere absolute Anzahl neu startender Teilnehmer zurückzuführen ist.

Bei CycloneDDS ist die Rediscovery in den kleineren Konstellationen (K1–K2) vergleichbar schnell wie die initiale Discovery. In den größeren Konstellationen (K3–K5)

jedoch führt die Kombination aus hoher Systemlast und der Wartezeitlogik für SPDP-Nachrichten zu deutlich erhöhten Latenzen.

Zusammenfassend zeigt sich, dass vSomeIP durchgängig eine robuste und zuverlässige Ausfallerkennung und Rediscovery bietet, während die Ergebnisse bei CycloneDDS gemischt sind.

6.5.4 Konfigurierbarkeit und Anpassungsfähigkeit

vSomeIP weist mehrere technisch ausgereifte Parameter zur detaillierten Anpassung der Service Discovery auf, von denen nur eine Teilmenge im Rahmen dieser Arbeit getestet werden konnte. Weitere quantitative Tests zu Parametern wie `repetitions_min` und `repetitions_max` wären lohnenswert.

Ein Effekt wie der von `initial_delay_{min, max}` in vSomeIP, bei dem eine Erhöhung in größeren Konstellationen die Latenz verringert, ist bei `SPDPResponseMaxDelay` in CycloneDDS nicht erkennbar. Dies deutet auf Unterschiede in der Ausgereiftheit der Konfigurationslogik beider Middlewares hin.

In dieser Arbeit wurden alle Teilnehmer jedes Testdurchlaufs mit den gleichen Konfigurationsparametern gestartet. So sind bei vSomeIP in manchen Durchläufen alle Teilnehmer zufällig zwischen 500 und 2000 ms gestartet. Alternativ könnten Teilnehmern unterschiedliche Werte für `initial_delay_{min, max}` zugewiesen werden, sodass kritische Dienste die Initial Wait Phase schneller verlassen und somit schneller verfügbar sind als weniger kritische. Dies würde eine noch flexiblere Anpassung an die spezifischen Anforderungen eines Automotive-Systems ermöglichen.

6.6 Herausforderungen und Limitationen

Die Ergebnisse dieser Messstudie sind durch verschiedene Faktoren limitiert, welche im Folgenden erläutert werden. In realen Automotive-Umgebungen sind Middleware-Anwendungen auf verschiedene ECUs verteilt. In der hier verwendeten Testumgebung laufen jedoch alle Teilnehmer auf einem einzigen Host und teilen sich gemeinsame Ressourcen (CPU, RAM, SSD), wodurch sich die Anwendungen gegenseitig beeinflussen können. *Salomon et al.* [28] weisen in ihrer Studie ebenfalls auf diese Limitation bei der

Nutzung von Mininet hin. Zwar lassen sich in Mininet verschiedene Netzwerktopologien mit Eigenschaften wie Bandbreite, Latenzen und Paketverlusten emulieren, jedoch ist die Emulation eines realen Automotive-Netzwerks in Hardware mit all seinen Eigenschaften nur bedingt möglich. Systembedingt nutzt Mininet einen einzelnen Linuxkernel für alle virtuellen Hosts. Es hat sich gezeigt, dass vor allem bei CycloneDDS die Leistungsfähigkeit der Emulation einen Flaschenhals dargestellt hat, weshalb die Ergebnisse in den größeren Konstellationen ab K3 mit Vorsicht zu interpretieren sind und recht sicher nicht die Performance in realen Automotive-Umgebungen widerspiegeln. Allerdings gibt die Testumgebung implizit Aufschluss darüber, wie ressourcenintensiv die jeweiligen Middleware-Stacks arbeiten und wie gut sie mit steigender Teilnehmeranzahl skalieren.

Darüber hinaus ist zu bedenken, dass die hier verwendeten Publisher- und Subscriberzahlen zwar realistische Szenarien abbilden, jedoch in realen Automotive-Umgebungen meist mehrere Publisher und Subscriber pro Anwendung existieren. Somit ist das 1-zu-1-Verhältnis von Node zu Anwendung in dieser Arbeit zwar nicht absolut realitätsgetreu, bildet aber eine geeignete Annäherung.

Die Szenarien S4 und S5 stellen mit 50 % Ausfall beide Extremfälle dar, welche in realen Automotive-Umgebungen höchstwahrscheinlich so nicht vorkommen würden. Sie sind eher exemplarischer Natur und daher zu Illustration von Recovery-Verhalten geeignet.

7 Fazit und Ausblick

Diese Arbeit hat eine quantitative Untersuchung hinsichtlich der Latenz und Effizienz der Service Discovery-Prokolle von SOME/IP und DDS in einer automatisierten Testumgebung auf Basis von Mininet durchgeführt. Hierbei wurden typische Teilnehmerkonstellationen und Szenarien aus dem Automobilbereich betrachtet und verschiedene Konfigurationsparameter variiert. Aus der Arbeit lassen sich folgende Erkenntnisse ableiten:

7.1 Erkenntnisse

vSomeIP skaliert nahezu linear mit der Teilnehmerzahl In allen getesteten Szenarien und Konstellationen erreicht vSomeIP kürzere Discovery-Zeiten als CycloneDDS, wobei die Latenz nahezu linear mit der Teilnehmerzahl steigt. Bei CycloneDDS hingegen ist dies aufgrund des Ressourcenbedarfs in der Testumgebung nicht hinreichend zu bewerten.

CycloneDDS erzeugt ein Vielfaches an Netzwerkverkehr Während vSomeIP in den größten Konstellationen etwas unter einem MiB an Paketen für die Service Discovery überträgt, generiert CycloneDDS teils das 200-fache nur an SPDP/SEDP-Nachrichten. Dies kann insbesondere in ressourcenbeschränkten Automotive-Umgebungen kritisch sein.

Der Ressourcenbedarf von CycloneDDS ist höher als der von vSomeIP CycloneDDS benötigt in größeren Konstellationen deutlich mehr CPU- und Arbeitsspeicherressourcen als vSomeIP, was den Einsatz auf ECUs mit begrenzter Rechenleistung erschweren kann.

Die Parameter in vSomeIP ermöglichen präzise Steuerung der Discovery-Phase Die Parameter `initial_delay_min` und `initial_delay_max` zeigen in größeren Topologien klare Optimierungspotenziale: während in K1–K3 ein sofortiger Start optimal ist, reduziert in K5 eine Streuung von 500–2000 ms die Latenz durch Vermeidung von Nachrichtenexplosionen. Bei CycloneDDS zeigen die getesteten Parameter nur in Extremkonstellationen erkennbare Effekte.

vSomeIP zeigt Robustheit bei Ausfällen in allen Konstellationen Die Rediscovery nach Teilnehmerausfall erfolgt in vSomeIP in beiden Neustart-Szenarien nahezu identisch zur initialen Discovery. Bei CycloneDDS hingegen ist die Latenz in in S4 (Subscriber-Neustart) deutlich höher als in S5 (Publisher-Neustart) und weist eine hohe Tail Latency auf, was wiederum auf Systemüberlastung hindeutet.

7.2 Ausblick

Diese Arbeit kann als Anknüpfungspunkt für weitere Untersuchungen zur Service Discovery beiden Middlewares dienen. Unter anderem könnten weitere Parameterkombinationen, Topologien und realitätsnähere Netzwerkbedingungen untersucht werden. So könnten beispielsweise Netzwerklatenzen, Paketverluste und Bandbreitenbeschränkungen in die Testumgebung integriert werden, um die Robustheit der Protokolle unter realistischeren Bedingungen zu testen. Hierbei könnte DDS durch sein verlässliches RTPS-Protokoll beispielsweise Vorteile aufweisen, die im Rahmen dieser Arbeit nicht betrachtet wurden.

Auch die Implementierungen der Middlewares könnten variiert werden; insbesondere für DDS gibt es weitere Open-Source-Implementierungen wie *FastDDS* oder *OpenDDS*, deren Ressourceneffizienz und Geschwindigkeit sich von CycloneDDS unterscheiden könnten. In diesem Kontext ist insbesondere der Standard DDS-XRCE [21] zu nennen, der speziell für ressourcenbeschränkte Geräte entwickelt wurde und daher im Automotive-Bereich von Interesse sein könnte. Hierbei wird ein Broker zwischen den ressourcenbeschränkten ECUs und dem DDS-Domain-Netzwerk eingesetzt, um die Komplexität und Rechenlast auf den Steuergeräten zu reduzieren.

Netzwerkemulatoren wie Mininet bieten zwar eine gute Möglichkeit, verschiedene Szenarien schnell und kostengünstig zu testen, jedoch sind die Ergebnisse nicht 1:1 auf rea-

le IVNs übertragbar. Daher könnte eine Testumgebung mit echter verteilter Hardware realistischere und praxisnahe Ergebnisse liefern. So könnte beispielsweise eine moderne zonale Architektur mit Time-Sensitive Networking-fähigen Switches als Testumgebung dienen, um die Service Discovery unter realistischen Netzwerkbedingungen zu evaluieren. Das für diese Arbeit entwickelte Evaluationsmodul arbeitet mit typischen pcapng-Dateien und kann daher auch losgelöst von Mininet mit Daten außerhalb der Emulation arbeiten. Dabei sind jedoch Zeitaspekte zu beachten: So müssen in einem verteilten System die Uhren der beteiligten Maschinen synchronisiert sein, um eine vergleichbare Genauigkeit wie in der Emulation zu gewährleisten. Weiterhin können Netzwerklatenzen die Messungen stark beeinflussen, da diese in der jetzigen Implementation nur auf der Empfängerseite durchgeführt werden.

Insgesamt kann die vorliegende Arbeit als Grundgerüst für weiterführende Arbeiten zur Quantifizierung und Optimierung der Service Discovery in Automotive-Middlewares dienen. Realitätsnahe Tests sind lohnenswert, um die gewonnenen Erkenntnisse zu validieren und geeignete Handlungsempfehlungen für den Einsatz in Software Defined Vehicles abzuleiten.

Literaturverzeichnis

[1] AUTOSAR: Requirements on SOME/IP Service Discovery Protocol. (2024), November, Nr. FO R24-11. – URL https://www.autosar.org/fileadmin/standards/R24-11/FO/AUTOSAR_FO_RS_SOMEIPServiceDiscoveryProtocol.pdf. – Zugriffsdatum: 2025-10-19

[2] AUTOSAR: *SOME/IP Protocol Specification FO*. November 2024. – URL https://www.autosar.org/fileadmin/standards/R24-11/FO/AUTOSAR_FO_PRS_SOMEIPProtocol.pdf. – Zugriffsdatum: 2025-02-28

[3] AUTOSAR: *SOME/IP Service Discovery Protocol Specification FO*. 2024. – URL https://www.autosar.org/fileadmin/standards/R24-11/FO/AUTOSAR_FO_PRS_SOMEIPServiceDiscoveryProtocol.pdf

[4] Bob Lantz and the Mininet Contributors: *Mininet/Mininet*. Mininet. November 2025. – URL https://github.com/mininet/mininet. – Zugriffsdatum: 2025-11-09

[5] Bode, Vincent ; Trinitis, Carsten ; Schulz, Martin ; Buettner, David ; Preclik, Tobias: DDS Implementations as Real-Time Middleware – A Systematic Evaluation. In: *2023 IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, URL https://ieeexplore.ieee.org/document/10296359/?arnumber=10296359. – Zugriffsdatum: 2025-03-27, August 2023, S. 186–195. – ISSN 2325-1301

[6] Bordoloi, Unmesh ; Chakraborty, Samarjit ; Jochim, Markus ; Joshi, Prachi ; Raghuraman, Arvind ; Ramesh, S.: Autonomy-Driven Emerging Directions in Software-defined Vehicles. In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, URL https://ieeexplore.ieee.org/document/10136910/?arnumber=10136910. – Zugriffsdatum: 2025-03-03, April 2023, S. 1–6. – ISSN 1558-1101

[7] Cakir, Mehmet ; Hackel, Timo ; Reider, Sandra ; Meyer, Philipp ; Korf, Franz ; Schmidt, Thomas C.: A QoS Aware Approach to Service-Oriented Communication in Future Automotive Networks. In: *2019 IEEE Vehicular Networking Conference (VNC)*. Los Angeles, CA, USA : IEEE, Dezember 2019, S. 1–8. – URL https://ieeexplore.ieee.org/document/9062794/. – Zugriffsdatum: 2025-03-03. – ISBN 978-1-7281-4571-6

[8] Connected Vehicle Systems Alliance: *vSomeIP Projektwebsite*. – URL https://covesa.global/project/vsomeip/. – Zugriffsdatum: 2025-09-25

[9] Connected Vehicle Systems Alliance: *vSomeIP Repository*. März 2025. – URL https://github.com/COVESA/vsomeip. – Zugriffsdatum: 2025-03-09

[10] Fraccaroli, Enrico ; Joshi, Prachi ; Xu, Shengjie ; Shazzad, Khaja ; Jochim, Markus ; Chakraborty, Samarjit: Timing Predictability for SOME/IP-based Service-Oriented Automotive In-Vehicle Networks. In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, April 2023, S. 1–6. – URL https://ieeexplore.ieee.org/document/10137065/?arnumber=10137065. – Zugriffsdatum: 2025-04-07. – ISSN 1558-1101

[11] Henle, Jacqueline ; Stoffel, Martin ; Schindewolf, Marc ; Nägele, Ann-Therese ; Sax, Eric: Architecture Platforms for Future Vehicles: A Comparison of ROS2 and Adaptive AUTOSAR. In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, URL https://ieeexplore.ieee.org/document/9921894/?arnumber=9921894. – Zugriffsdatum: 2025-04-03, Oktober 2022, S. 3095–3102

[12] Hunter, J. D.: Matplotlib: A 2D Graphics Environment. In: *Computing in Science & Engineering* 9 (2007), Nr. 3, S. 90–95

[13] IEEE: *Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks*. Dezember 2022. – URL https://ieeexplore.ieee.org/document/10004498/. – Zugriffsdatum: 2025-03-03

[14] Klüner, David P. ; Hegerath, Lucas ; Hatib, Amin D. ; Kowalewski, Stefan ; Alrifaee, Bassam ; Kampmann, Alexandru: *Automotive Middleware Performance: Comparison of FastDDS, Zenoh and vSomeIP*. Mai 2025. – URL http://arxiv.org/abs/2505.02734. – Zugriffsdatum: 2025-09-17

[15] Klüner, David P. ; Molz, Marius ; Kampmann, Alexandru ; Kowalewski, Stefan ; Alrifaee, Bassam: *Modern Middlewares for Automated Vehicles: A Tutorial*. Dezember 2024. – URL http://arxiv.org/abs/2412.07817. – Zugriffsdatum: 2025-04-03

[16] Mininet Contributors: *Mininet Overview*. – URL https://mininet.org/overview/. – Zugriffsdatum: 2025-11-09

[17] Muelas, David ; Ramos, Javier ; Vergara, Jorge E. L. de: Assessing the Limits of Mininet-Based Environments for Network Experimentation. In: *IEEE Network* 32 (2018), November, Nr. 6, S. 168–176. – URL https://ieeexplore.ieee.org/abstract/document/8515678. – Zugriffsdatum: 2025-11-10. – ISSN 1558-156X

[18] Nayak, Naresh ; Ambalavanan, Uthra ; Thampan, Jishnu M. ; Grewe, Dennis ; Wagner, Marco ; Schildt, Sebastian ; Ott, Jörg: Reimagining Automotive Service-Oriented Communication: A Case Study on Programmable Data Planes. In: *IEEE Vehicular Technology Magazine* 18 (2023), Juni, Nr. 2, S. 69–79. – URL https://ieeexplore.ieee.org/document/10017296/?arnumber=10017296. – Zugriffsdatum: 2025-04-07. – ISSN 1556-6080

[19] Object Management Group (OMG): *Appendix F: Glossary of Terms Related to DDS*. – URL https://www.omgwiki.org/ddsf/doku.php?id=ddsf:public:guidebook:06_append:glossary:start. – Zugriffsdatum: 2025-11-08

[20] Object Management Group (OMG): *What Is DDS?*. – URL https://www.dds-foundation.org/what-is-dds-3/. – Zugriffsdatum: 2025-09-29

[21] Object Management Group (OMG): *DDS for eXtremely Resource Constrained Environments (DDS-XRCE)*. Februar 2020. – URL https://www.omg.org/spec/DDS-XRCE/About-DDS-XRCE/. – Zugriffsdatum: 2025-11-09

[22] Object Management Group (OMG): *DDS Interoperability Wire Protocol (DDSI-RTPS) 2.5*. April 2022. – URL https://www.omg.org/spec/DDSI-RTPS/2.5/PDF. – Zugriffsdatum: 2025-09-25

[23] Open Robotics: *Eclipse Cyclone DDS — ROS 2 Documentation: Kilted Documentation*. – URL https://docs.ros.org/en/kilted/Installation/RMW-Implementations/DDS-Implementations/Working-with-Eclipse-CycloneDDS.html. – Zugriffsdatum: 2025-09-26

[24] Real-Time Innovations: *RTI Announces DDS Adoption into All AUTOSAR Packages and Platforms*. Dezember 2024. – URL https://www.rti.com/news/rti-announces-dds-adoption-into-all-autosar-packages. – Zugriffsdatum: 2025-09-25

[25] Roques, Arnaud ; PlantUML Contributors: *PlantUML Software*. – URL https://github.com/plantuml/plantuml

[26] Rumez, Marcel ; Grimm, Daniel ; Kriesten, Reiner ; Sax, Eric: An Overview of Automotive Service-Oriented Architectures and Implications for Security Countermeasures. In: *IEEE Access* 8 (2020), S. 221852–221870. – URL https://ieeexplore.ieee.org/document/9285284/?arnumber=9285284. – Zugriffsdatum: 2025-03-03. – ISSN 2169-3536

[27] Salomon, Timo ; Maile, Lisa ; Meyer, Philipp ; Korf, Franz ; Schmidt, Thomas C.: *Negotiating Strict Latency Limits for Dynamic Real-Time Services in Vehicular Time-Sensitive Networks*. Juli 2025. – URL http://arxiv.org/abs/2504.05793. – Zugriffsdatum: 2025-08-17

[28] Salomon, Timo ; Mueller, Mehmet ; Meyer, Philipp ; Schmidt, Thomas C.: *Building Automotive Security on Internet Standards: An Integration of DNSSEC, DANE, and DANCE to Authenticate and Authorize In-Car Services*. Juni 2025. – URL http://arxiv.org/abs/2506.13261. – Zugriffsdatum: 2025-10-23

[29] Sciangula, Gerlando ; Casini, Daniel ; Biondi, Alessandro ; Scordino, Claudio ; Di Natale, Marco: Bounding the Data-Delivery Latency of DDS Messages in Real-Time Applications. In: *LIPIcs, Volume 262, ECRTS 2023* 262 (2023), S. 9:1–9:26. – URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2023.9. – Zugriffsdatum: 2025-10-18. – ISBN 9783959772808

[30] Seyler, Jan R. ; Streichert, Thilo ; Glass, Michael ; Navet, Nicolas ; Teich, Jürgen: Formal Analysis of the Startup Delay of SOME/IP Service Discovery. In: *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, URL https://ieeexplore.ieee.org/document/7092357/. – Zugriffsdatum: 2025-10-18, März 2015, S. 49–54. – ISSN 1558-1101

[31] Vetter, Andreas ; Obergfell, Philipp ; Guissouma, Houssem ; Grimm, Daniel ; Rumez, Marcel ; Sax, Eric: Development Processes in Automotive Service-oriented Architectures. In: *2020 9th Mediterranean Conference on Embedded Computing*

(MECO), URL https://ieeexplore.ieee.org/document/9134175/. – Zugriffsdatum: 2025-09-18, Juni 2020, S. 1–7. – ISSN 2637-9511

A Messergebnisse

Notationshinweise

Im Folgenden werden in Tabellen und Diagrammen Abkürzungen für Konfigurationsparameter verwendet, um die Darstellung kompakt zu halten.

vSomeIP-Parameter:

COD: `cyclic_offer_delay`

IDmin: `initial_delay_min`

IDmax: `initial_delay_max`

TTL: `ttl`

CycloneDDS-Parameter:

HBI: `HeartbeatInterval`

LD: `LeaseDuration`

SRMD: `SPDPResponseMaxDelay`

DLPD: `DiscoveredLocatorPruneDelay`

Szenario-spezifische Parameter:

DTE: `downtime_exceed`: Gibt an, ob in den Szenarien 4 u. 5 die Ausfallzeit die `LeaseDuration` (DDS) bzw. `ttl` (SOME/IP) überschritten hat.

A.1 Messergebnisse für Szenario S1

Abb. A.1: *CycloneDDS:* Mittlere Systemlatenz in Szenario S1 je Konfiguration und Teilnehmerkonstellation

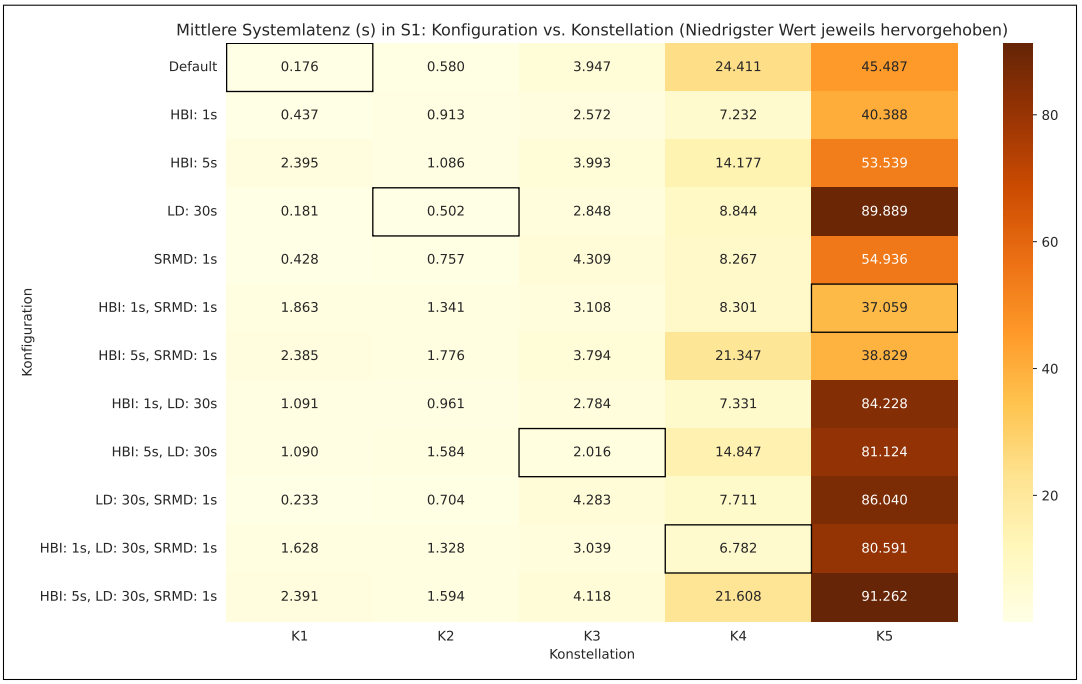
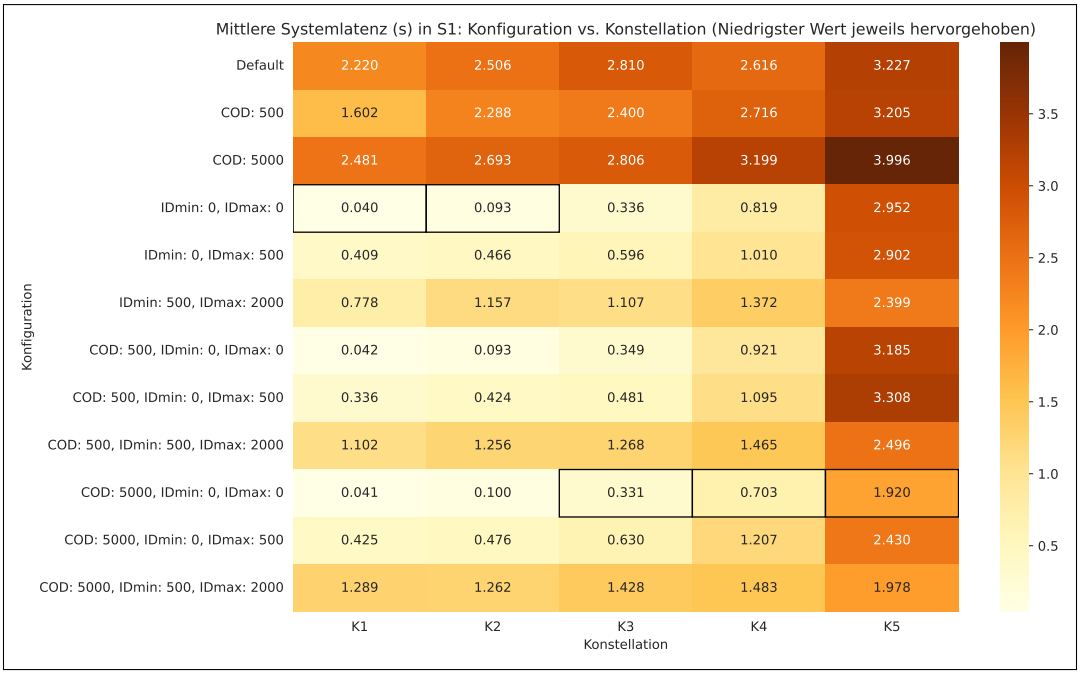


Abb. A.2: *vSomeIP:* Mittlere Systemlatenz in Szenario S1 je Konfiguration und Teilnehmerkonstellation



Tab. A.1: Schnellste Testdurchläufe nach Konstellation für Szenario 1

Konst.	Stack	Parameter	Δt_{total} [s]	Δt_{Pubs} [s]	Δt_{Subs} [s]	Pakete
K1	CycloneDDS	HBI: 1s	0,0938	0,0885	0,0307	1,36 MiB
	vSomeIP	COD: 500, IDmin: 0, IDmax: 0	0,0310	0,0305	0,0303	29,35 KiB
K2	CycloneDDS	HBI: 5s, LD: 30s	0,3644	0,3485	0,0321	5,25 MiB
	vSomeIP	IDmin: 0, IDmax: 0	0,0903	0,0836	0,0903	62,34 KiB
K3	CycloneDDS	HBI: 5s, LD: 30s	1,3661	1,3122	0,2182	17,41 MiB
	vSomeIP	COD: 5000, IDmin: 0, IDmax: 0	0,3262	0,2942	0,3262	115,63 KiB
K4	CycloneDDS	HBI: 1s, LD: 30s, SRMD: 1s	6,7816	5,1091	6,7816	102,93 MiB
	vSomeIP	COD: 5000, IDmin: 0, IDmax: 0	0,6725	0,5532	0,6725	154,96 KiB
K5	CycloneDDS	HBI: 5s, SRMD: 1s	38,8290	38,1240	31,9223	392,87 MiB
	vSomeIP	COD: 5000, IDmin: 500, IDmax: 2000	1,6920	1,6813	1,6920	542,57 KiB

A.2 Messergebnisse für Szenario S2

Abb. A.3: *CycloneDDS:* Mittlere Systemlatenz in Szenario S2 je Konfiguration und Teilnehmerkonstellation

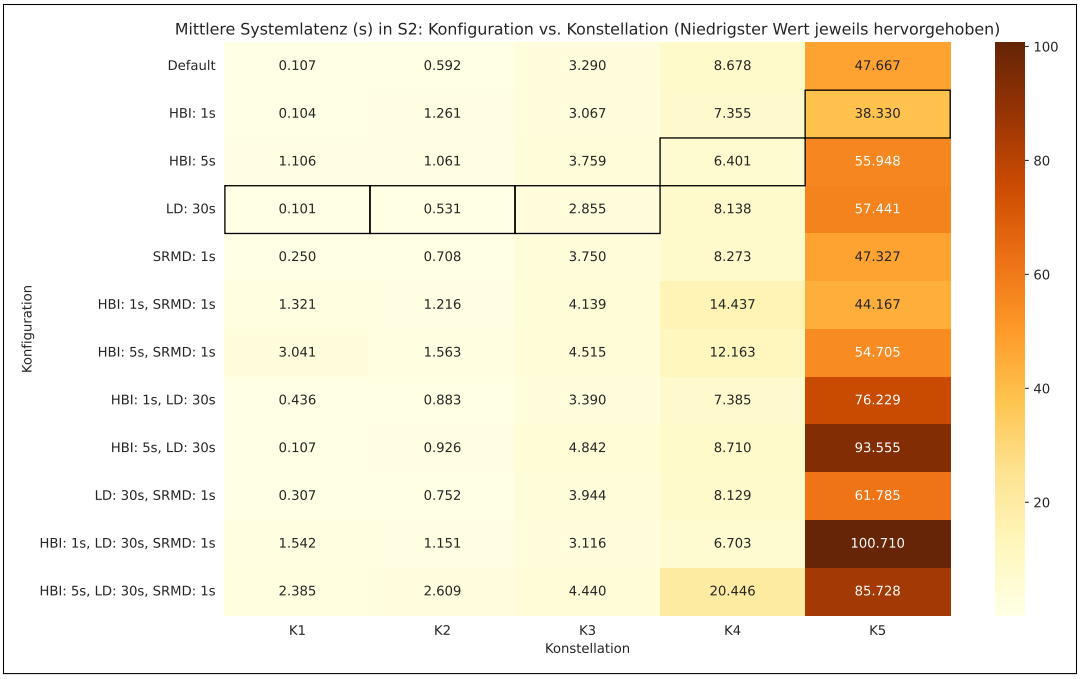
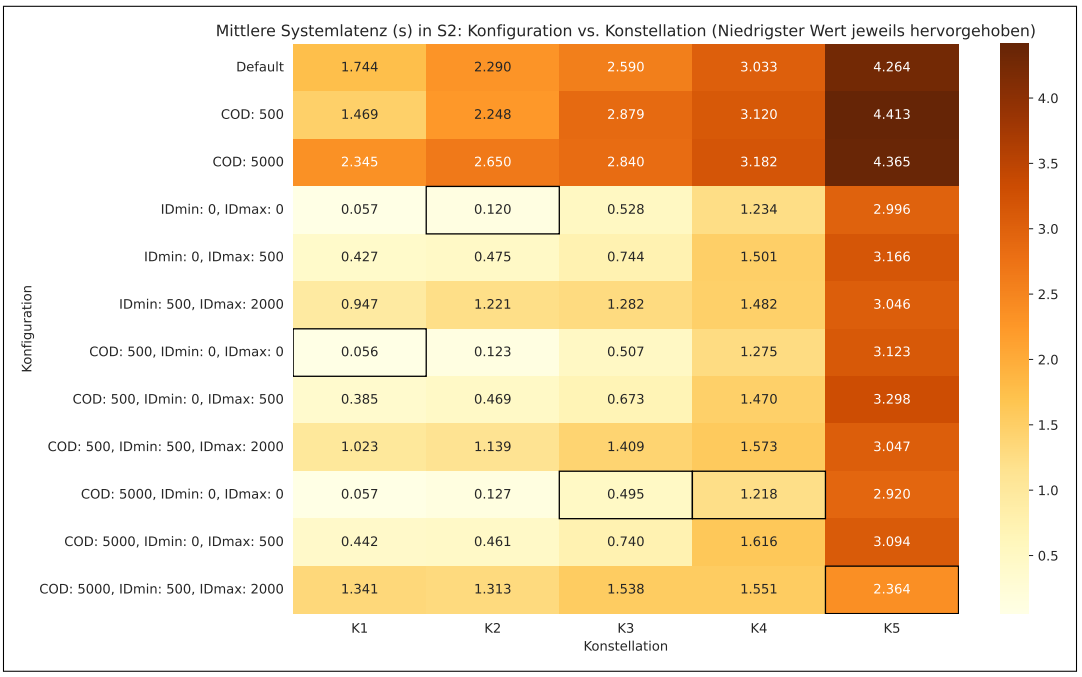


Abb. A.4: *vSomeIP:* Mittlere Systemlatenz in Szenario S2 je Konfiguration und Teilnehmerkonstellation



Tab. A.2: Schnellste Testdurchläufe nach Konstellation für Szenario 2

Konst.	Stack	Parameter	Δt_{total} [s]	Δt_{Pubs} [s]	Δt_{Subs} [s]	Pakete
K1	CycloneDDS	HBI: 1s	0,0789	0,0743	0,0335	1,21 MiB
	vSomeIP	COD: 500, IDmin: 0, IDmax: 0	0,0479	0,0472	0,0167	26,99 KiB
K2	CycloneDDS	Default	0,4568	0,4399	0,2614	4,83 MiB
	vSomeIP	IDmin: 0, IDmax: 0	0,1097	0,1093	0,0356	68,83 KiB
K3	CycloneDDS	HBI: 1s	2,1025	1,9345	1,2185	25,29 MiB
	vSomeIP	COD: 5000, IDmin: 0, IDmax: 0	0,4591	0,4506	0,1545	201,88 KiB
K4	CycloneDDS	HBI: 5s, LD: 30s	4,3552	4,3552	2,0036	67,83 MiB
	vSomeIP	IDmin: 500, IDmax: 2000	1,4125	1,4105	1,4113	352,09 KiB
K5	CycloneDDS	LD: 30s, SRMD: 1s	19,6459	9,9410	10,9147	122,13 MiB
	vSomeIP	COD: 5000, IDmin: 500, IDmax: 2000	2,2763	2,2738	2,2720	655,6 KiB

A.3 Messergebnisse für Szenario S3

Abb. A.5: *CycloneDDS:* Mittlere Systemlatenz in Szenario S3 je Konfiguration und Teilnehmerkonstellation

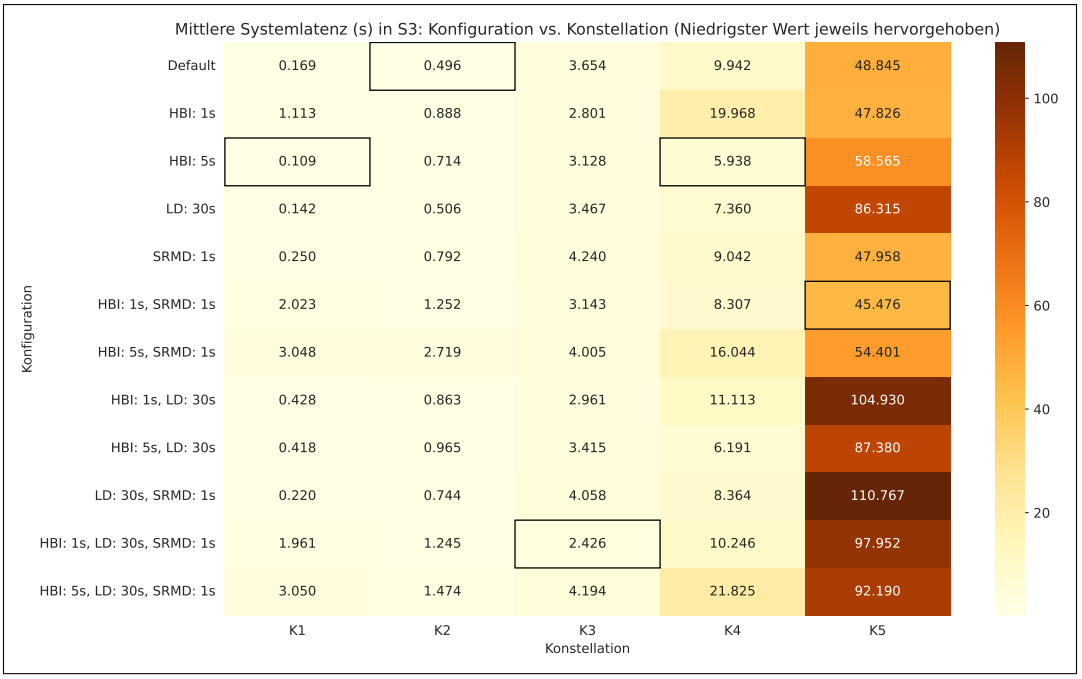
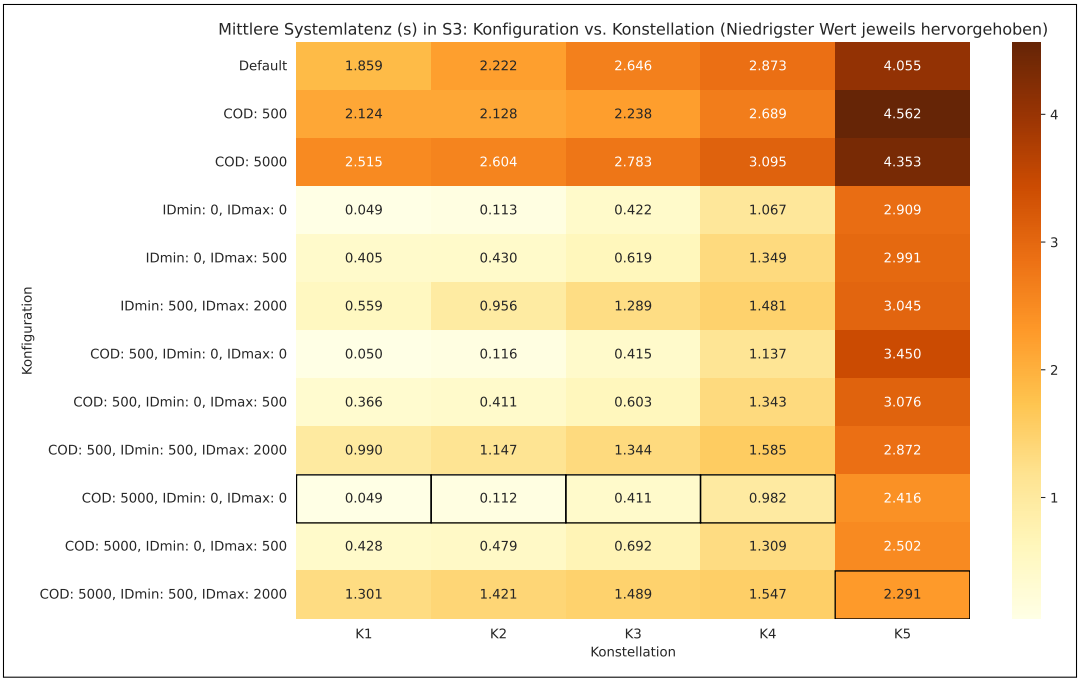


Abb. A.6: *vSomeIP:* Mittlere Systemlatenz in Szenario S3 je Konfiguration und Teilnehmerkonstellation



Tab. A.3: Schnellste Testdurchläufe nach Konstellation für Szenario 3

Konst.	Stack	Parameter	Δt_{total} [s]	Δt_{Pubs} [s]	Δt_{Subs} [s]	Pakete
K1	CycloneDDS	HBI: 1s, LD: 30s	0,0915	0,0915	0,0505	1,34 MiB
	vSomeIP	COD: 5000, IDmin: 0, IDmax: 0	0,0410	0,0408	0,0386	26,51 KiB
K2	CycloneDDS	HBI: 5s, LD: 30s	0,3989	0,3989	0,3926	5,85 MiB
	vSomeIP	IDmin: 0, IDmax: 0	0,1054	0,1052	0,0976	69,12 KiB
K3	CycloneDDS	HBI: 5s	1,9294	1,9294	1,0975	25,05 MiB
	vSomeIP	COD: 5000, IDmin: 0, IDmax: 0	0,3934	0,3932	0,3744	174,81 KiB
K4	CycloneDDS	HBI: 5s	5,9378	4,9452	5,9354	86,96 MiB
	vSomeIP	COD: 5000, IDmin: 0, IDmax: 0	0,9653	0,9651	0,9619	296,42 KiB
K5	CycloneDDS	HBI: 1s, SRMD: 1s	45,4830	45,4830	40,7601	370,46 MiB
	vSomeIP	COD: 5000, IDmin: 500, IDmax: 2000	2,1329	2,1230	2,1329	620,04 KiB

A.4 Messergebnisse für Szenario S4

Abb. A.7: *CycloneDDS:* Mittlere Systemlatenz in Szenario S4 je Konfiguration und Teilnehmerkonstellation

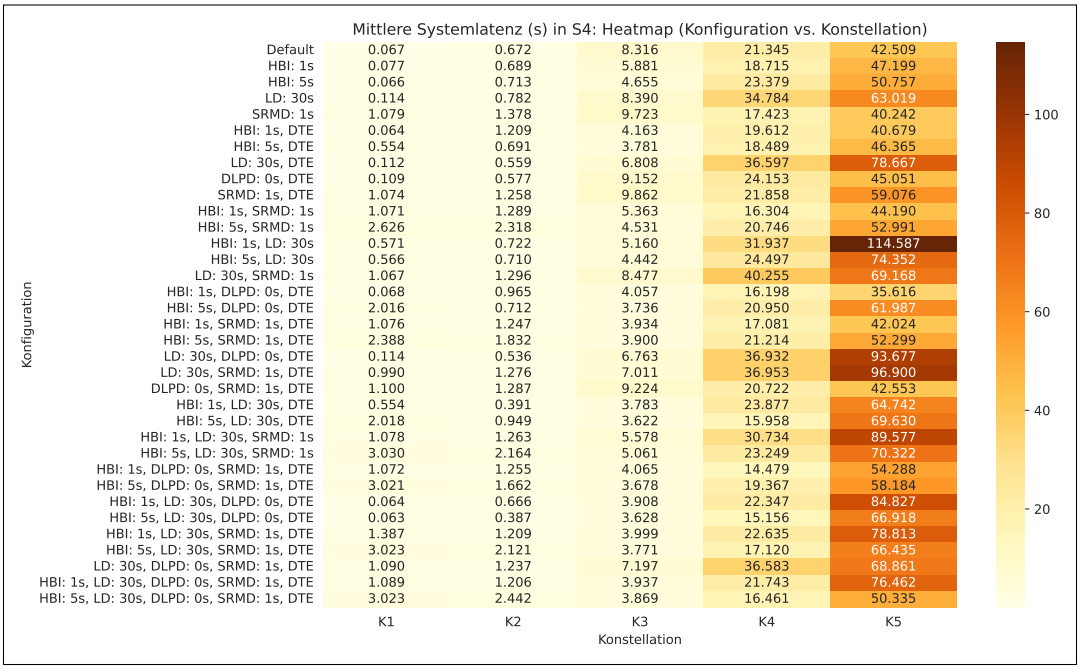
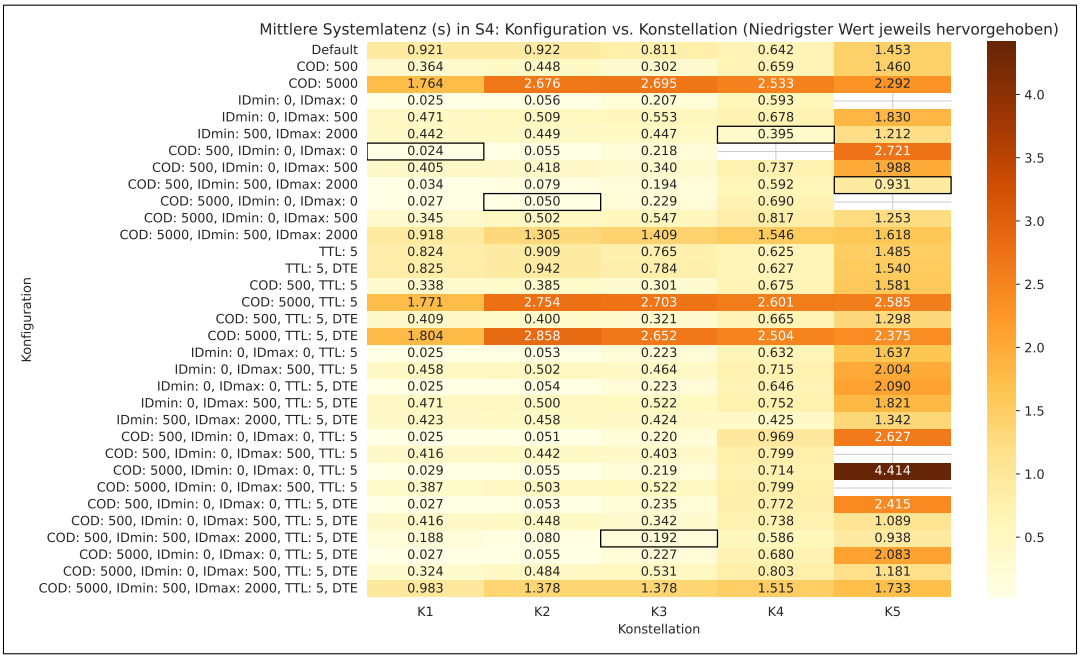


Abb. A.8: *vSomeIP:* Mittlere Systemlatenz in Szenario S4 je Konfiguration und Teilnehmerkonstellation



Tab. A.4: Schnellste Testdurchläufe nach Konstellation für Szenario 4

Konst.	Stack	Parameter	Δt_{total} [s]	Δt_{Pubs} [s]	Δt_{Subs} [s]	Pakete
K1	CycloneDDS	LD: 30s, DLPD: 0s, DTE	0,0606	0,0515	0,0588	923,0 KiB
	vSomeIP	IDmin: 0, IDmax: 0, TTL: 5, DTE	0,0201	0,0198	0,0193	8,82 KiB
K2	CycloneDDS	HBI: 5s, LD: 30s, DTE	0,3503	0,2983	0,3410	3,99 MiB
	vSomeIP	COD: 5000, IDmin: 0, IDmax: 0	0,0497	0,0494	0,0488	17,88 KiB
K3	CycloneDDS	HBI: 5s, LD: 30s, DTE	3,5853	3,0342	3,5459	36,87 MiB
	vSomeIP	COD: 500, IDmin: 500, IDmax: 2000, TTL: 5, DTE	0,1860	0,1854	0,1860	40,53 KiB
K4	CycloneDDS	HBI: 5s, LD: 30s, DLPD: 0s, DTE	15,1561	8,5398	14,9224	103,65 MiB
	vSomeIP	IDmin: 500, IDmax: 2000	0,3952	0,3947	0,3907	80,37 KiB
K5	CycloneDDS	HBI: 1s, DLPD: 0s, DTE	35,6155	35,6155	28,4232	213,12 MiB
	vSomeIP	COD: 500, IDmin: 500, IDmax: 2000	0,9313	0,9248	0,9313	159,2 KiB

A.5 Messergebnisse für Szenario S5

Abb. A.9: *CycloneDDS:* Mittlere Systemlatenz in Szenario S5 je Konfiguration und Teilnehmerkonstellation

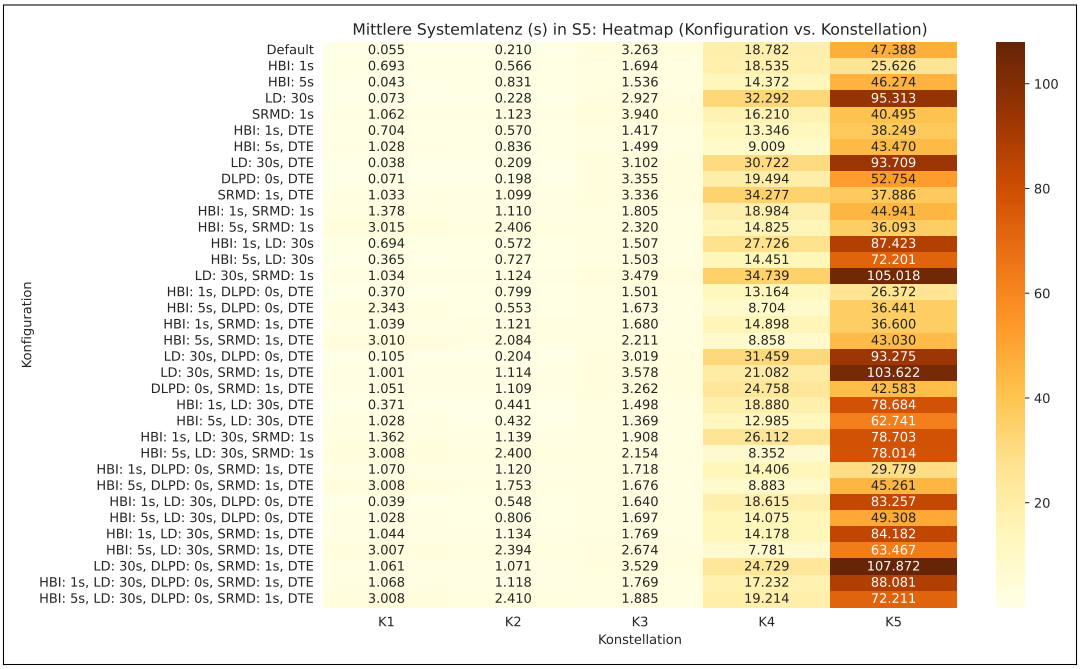
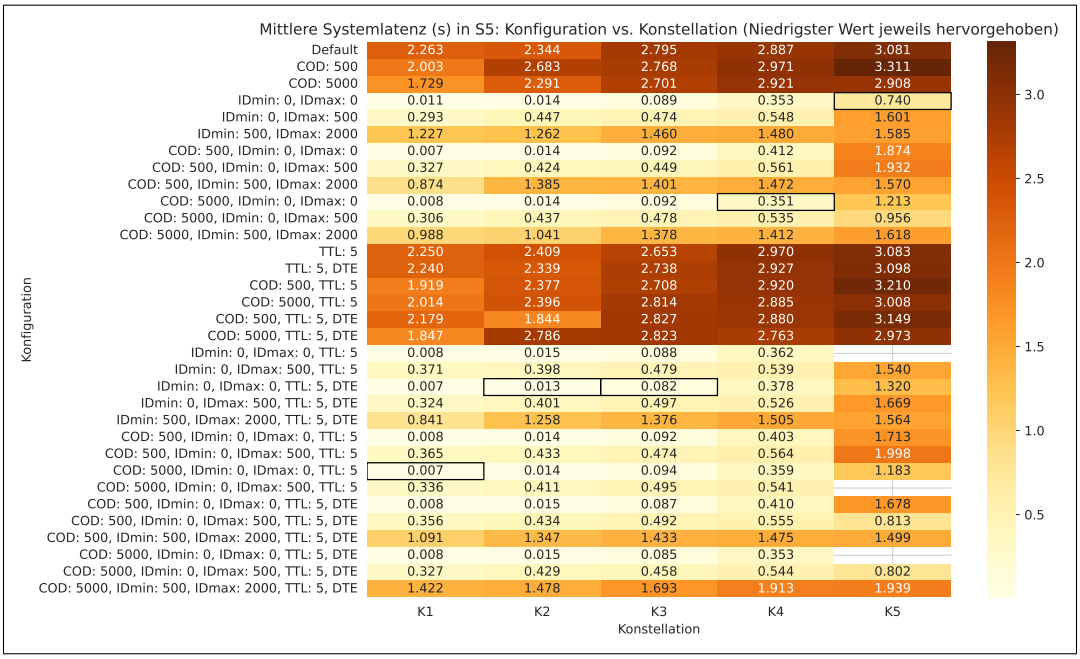


Abb. A.10: *vSomeIP:* Mittlere Systemlatenz in Szenario S5 je Konfiguration und Teilnehmerkonstellation



Tab. A.5: Schnellste Testdurchläufe nach Konstellation für Szenario 5

Konst.	Stack	Parameter	Δt_{total} [s]	Δt_{Pubs} [s]	Δt_{Subs} [s]	Pakete
K1	CycloneDDS	HBI: 1s, LD: 30s	0,0340	0,0321	0,0204	265,75 KiB
	vSomeIP	COD: 500, IDmin: 0, IDmax: 0, TTL: 5, DTE	0,0062	0,0014	0,0062	4,88 KiB
K2	CycloneDDS	HBI: 5s	0,1768	0,1715	0,0888	1,28 MiB
	vSomeIP	IDmin: 0, IDmax: 0, TTL: 5, DTE	0,0120	0,0076	0,0120	9,68 KiB
K3	CycloneDDS	HBI: 5s, LD: 30s, DTE	1,2333	1,2040	1,2333	6,09 MiB
	vSomeIP	COD: 500, IDmin: 0, IDmax: 0	0,0765	0,0653	0,0765	53,16 KiB
K4	CycloneDDS	HBI: 5s, LD: 30s, SRMD: 1s, DTE	7,6480	7,1909	6,6986	43,78 MiB
	vSomeIP	COD: 5000, IDmin: 0, IDmax: 0, TTL: 5, DTE	0,3330	0,3153	0,3330	131,11 KiB
K5	CycloneDDS	HBI: 1s	25,6260	24,8765	25,3314	154,61 MiB
	vSomeIP	COD: 5000, IDmin: 0, IDmax: 500, TTL: 5, DTE	0,6467	0,6331	0,6467	227,19 KiB

Glossar

E/E Der Begriff E/E-Architektur steht für elektrische/elektronische Architektur und umfasst die Gesamtheit der elektrischen und elektronischen Systeme in einem Fahrzeug.

Feldbus Ein Feldbus ist ein Bussystem, das in einer Anlage Feldgeräte wie Messfühler (Sensoren) und Stellglieder (Aktoren) zur Kommunikation mit einem Automatisierungsgerät verbindet.

Locator Bezeichnung für eine Netzwerkadresse im RTPS-Kontext, die angibt, wie ein Endpunkt (Endpoint) erreichbar ist.

Middleware Eine Middleware ist eine Software, mit der Anwendungen oder Komponenten in einem verteilten Netzwerk auf eine oder mehrere Arten kommunizieren können, und die somit eine Brücke zwischen Anwendungen und Betriebssystemen bildet.

Open vSwitch Ein Open-Source-Software-Switch, der erweiterte Netzwerkfunktionen für virtuelle Umgebungen bietet und in Mininet verwendet wird.

pcapng pcapng (Packet Capture Next Generation) ist ein Dateiformat zur Speicherung von Netzwerkverkehrsdaten, das erweiterte Funktionen im Vergleich zum älteren pcap-Format bietet, wie z.B. die Unterstützung mehrerer Schnittstellen und Metadaten.

Security Beschreibt im Englischen den Aspekt der Informationssicherheit, d.h. den Schutz von Daten.

Single Point of Failure Ein Bestandteil eines technischen Systems, dessen Ausfall den Ausfall des gesamten Systems nach sich zieht.

Tail Latency (Long) Tail Latency bezeichnet den unverhältnismäßig hohen Einfluss einer kleinen Anzahl von Netzwerkanfragen, deren Bearbeitung deutlich länger dauert als die der meisten anderen Anfragen und welche dadurch das Gesamtsystem verlangsamen.

Time-Sensitive Networking Time-Sensitive Networking (TSN) ist eine Sammlung von IEEE-Standards, die Erweiterungen für Ethernet-Netzwerke definieren, um zeitkritische Datenübertragung mit garantierter Latenz und Hochverfügbarkeit zu ermöglichen.

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original