UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Vergleichsanalyse von Kommunikationsprotokollen am Beispiel der Logistikdomäne Ostseeraum

Comparative analysis of communication protocols at the example of the Baltic Seas logistics domain

Bachelorarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Jan Niklas Pudschun

ausgegeben und betreut von
Prof. Dr. Martin Leucker

mit Unterstützung von
Tobias Braun

Lübeck, den 31. Juli 2020

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne die Benutzung anderer als der angegebenen Hilfsmittel selbständig verfasst habe; die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe des Literaturzitats gekennzeichnet.

(Jan Niklas Pudschun)
Lübeck, den 31. Juli 2020

Kurzfassung Diese Arbeit gibt einen Überblick über bekannte Kommunikationsprotokolle und führt eine Vergleichsanalyse für den Anwendungsfall einer Logistikplattform in der Logistikdomäne Ostseeraum durch. Die Ergebnisse der theoretischen Analyse werden durch praktische Tests überprüft. Auf Basis dieser Ergebnisse wird am Ende eine Empfehlung über das am besten geeignete Protokoll im Rahmen einer solchen Logistikplattform, abgeben.

Abstract This thesis gives an overview of known communication protocols and carries out a comparative ananlysis for the use case of a logistics platform inside the logistics domain baltic sea. The results of the theoretical analysis will be examined through practical tests. Based upon the results a recommendation for the best suited protocol in context of such a logistics platform, will be given at the end.

Inhaltsverzeichnis

1. Einleitung

Mit wachsender Rechenkraft von Computern und stärkeren Vernetzung eben dieser steigen die Möglichkeiten zur Digitalisierung und Automatisierung von Prozessen. Dadurch steigen die Möglichkeiten und Optionen für das Internet of Things (IoT). Gerade in der Logistik können dadurch neue Wege gegangen werden und die Verwaltung und Überwachung vieler verschiedener Transportwege,-mittel und –prozesse optimiert werden. Das Vorankommen dieser Entwicklung kann gerade durch Verwaltungsprogramme für Umschlagzentren wie Häfen unterstützt und getragen werden. So zum Beispiel auch das RoRo Hafen 4.0. Projekt am Institut für Softwareengineering und Programmiersprachen der Universität Lübeck. Dieses Projekt befindet sich noch im Stadium einer Machbarkeitsstudie und für eine Umsetzung einer solchen Softwareplattform stellt sich die Frage, wie die einzelnen Komponenten der Transport und Logistikketten miteinander kommunizieren sollen. Diese Arbeit beschäftigt sich mit diesem Teil und genauer damit mit Hilfe welcher Protokolle diese Kommunikation erfolgen soll. Da sich für diese Nutzung ein asynchrones Kommunikationsmodell anbietet, wird sich genauer mit den Optionen für ein System nach dem Publish/Subscribe(Pub/Sub)-Modell beschäftigt.

1.1. Verwandte Arbeiten

Da dieser Bereich weder ganz neu ist, noch dies das einzige Projekt ist, das sich mit den Optionen des IoT beschäftigt, gibt auch schon ein umfangreiche Menge an Forschung zu dem Thema. So gibt es Studien die sich bereits damit beschäftigen welche Anforderungen die Middleware für eine solche Lösung erfüllen muss[HKM+16]. Allerdings wurde dort eine recht begrenzte Anzahl verschiedener Protokolle für ihre Eignung in einer IoT Umgebung analysiert und für einen Anwendungsfall der auf dem RoRo 4.0 Projekt basiert, fehlt dort ein Überblick über die Landschaft der verfügbaren Protokolle, sowie eine Analyse dieser Protokolle im Rahmen der Logistikdomäne Ostseeraum. Dies soll im Rahmen dieser Arbeit durch eine Vergleichsanalyse ausgewählter Protokolle erreicht werden, um am Ende eine Entscheidungshilfe zu geben und eine subjektive Empfehlung aussprechen zu können.Ähnliche Untersuchungen gibt es bereits in mehreren Studien. So gibt es bereits einen Vergleich beliebter Messaging Protokolle aus dem Vereinten Königreich[Nai17], allerdings fehlen dort noch Protokolle in der Analyse und der Bezug auf den Anwendungsfall

dieser Arbeit, genauso gab es auch bereits eine Performance Analyse verschiedener Protokolle[CK16]. Dort werden wenige Protokolle auf Zuverlässigkeit und Latenz geprüft um ihre Eignung für z.B. den medizinischen Sektor auszuloten. Es existiert bereits eine Analyse von Webanwendungen verschiedener Protokolle[BPM16] für Echtzeitsysteme. Da eine Hafenverwaltung aber nur sehr begrenzte Echtzeitanforderungen erfüllen muss, ist dieser Fokus für diese Arbeit nicht voll zutreffend. Auch werden diese Kommunikationsprotokolle für andere Anwendungsfälle wie autonomes Fahren analysiert[KRD11], was aufgrund der sehr unterschiedlichen Anforderungen in unterschiedlichen IoT Netzen nur sehr begrenzt hilft. Des weiteren wurde sich auch bereits mit den Grenzen des des Pub/Sub Patterns in IoT Anwendungen beschäftigt[HW16]. Jedoch ist nach besten Wissen und Gewissen diese Arbeit die Einzige, welche einen Überblick und Vergleich so vieler Protokolle darstellt und diese Protokolle anhand ihrer Eignung für Softwarelösungen zur Verwaltung der Logistikdomäne Ostseeraum bewertet.

1.2. Aufbau der Arbeit

In Kapitel 2.1. wird erklärt, was Kommunikationsprotokolle sind und welche Attribute Sie ausmachen. Es werden in 2.1.1. alle Protokolle, die analysiert werden, in Ihrer Funktionsweise und Charakteristik vorgestellt. Im Part 2.2. wird gezeigt wie sich die Logistikdomäne Ostseeraum im Rahmen dieser Arbeit definiert und welcher Anwendungsfall für die Analyse sich daraus ergibt. In Kapitel 3.1. wird zuerst definiert anhand welcher Attribute die Vergleichsanalyse durchgeführt wird. Danach wird in Abschnitt 3.2. dargestellt, wie sich die Bewertungsmatrix für die Analyse aufbaut und wie sie die Logistikdomäne Ostseeraum widerspiegelt. Der Abschnitt 3.3. stellt die Ergebnisse der Vergleichsanalyse dar;gefolgt von Teil 3.4., der die Erklärung und Methodik der praktischen Überprüfung der Ergebnisse darlegt. In Kapitel 4 werden dann die Ergebnisse der Vergleichsanalyse und der Praxistests ausgewertet und diskutiert. Schlussendlich wird Kapitel 5 die subjektive Empfehlung für den hier betrachteten Anwendungsfall gegeben und Hinweise auf eventuell notwendige weitere Untersuchungen genannt.

2. Grundlagen

Die Kommunikation zwischen Computern kann als Austausch von Datenpaketen angesehen werden[Sha08]. Dieser Austausch geschieht in kleinen, diskreten, elementaren Schritten. Diese Schritte sind die Nachrichten, die verschickt werden[Sha08]. Diese Nachrichten können sehr verschiedene Inhalte haben, von einzelnen elektronischen Signalen bis hin zu größeren und komplexeren Daten[Sha08]. Da beide Parteien aber wissen müssen, wie diese Nachrichten aufgebaut sind eventuell auch von wem Sie kommen, braucht es wie in der zwischenmenschlichen Kommunikation auch zwischen Computern Regeln und Strukturen, um einen Nachrichtenaustausch zu ermöglichen.
Im Internet werden diese Regeln durch die Internet Protokoll Suite und das TCP/IP Modell verwirklicht. Diese sind seit sie im Oktober 1989 in den RFC 1122[Bra89] und RFC 1123[Bra] von der Internet Engineering Task Force definiert wurden, der Standard im Internet. Für diese Arbeit sind die Protokolle der Anwendungsschicht relevant, da untersucht werden soll ,welche Protokolle in der Kommunikation zwischen Anwendungen welche Unterschiede haben und wie sich diese auswirken. Ebenso wird davon ausgegangenen, dass eine Verbindung über das Internet mithilfe des TCP/IP Protokollstacks zwischen Gesprächspartner möglich ist und für die Kommunikation gewählt wird.

2.1. Kommunikationsprotokolle

Die Kommunikationsprotokolle der Anwendungsschicht definieren die Nachrichtenstruktur gesendeter Daten, in welchem Format diese abgespeichert werden und welche Optionen für sicheren und zuverlässigen Datenaustausch vorhanden sind. Sie regeln allerdings auch, wie sich Anwendungen identifizieren müssen und was bei Nachrichtenverlust oder ungeplantem Verbindungsabbruch passiert. Protokolle regeln des weiteren, welche Nachrichten überhaupt gesendet werden dürfen und wie auf bestimmte Nachrichten geantwortet werden muss. Dabei können sie von andern Protokollen des Protokollstapels unterstützt werden, etwa durch die Verwendung des TCP Protokolls in der Transportschicht, da dieses gewisse Sicherheits- und Zuverlässigkeitsstandards für eine Verbindung definiert. Kommunikationsprotokolle lassen sich auf Grund sehr viele verschieden Eigenschaften definieren und

vergleichen. Zuerst wird definiert, über welche Eigenschaften sich Kommunikationsprotokolle beschreiben und unterscheiden lassen. Die Auswahl basiert auf in der Recherche besonders häufig auftretenden Themen und Schlagwörtern. Zur besseren Visualisierung wurden die Ergebnisse in Abbildung 2.1 dargestellt.

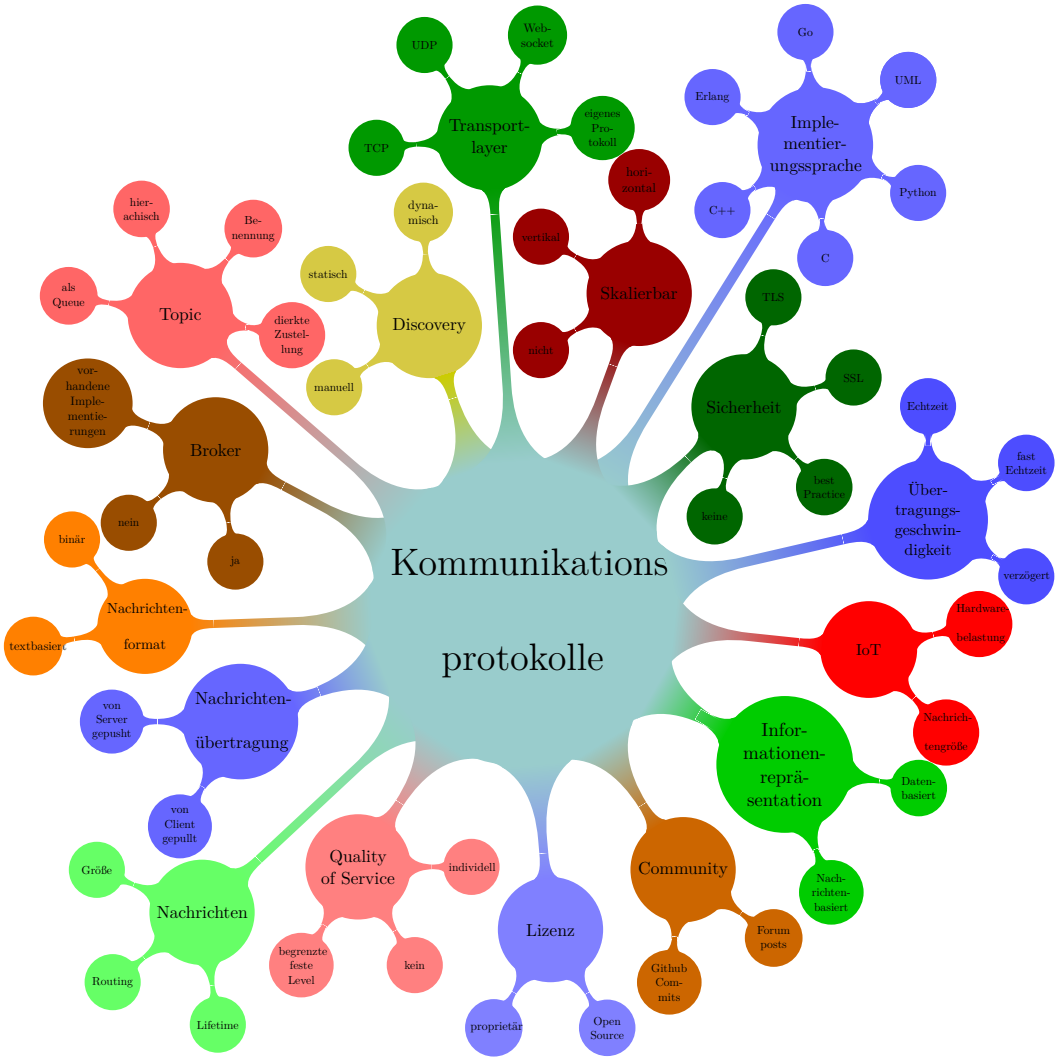

Abbildung 2.1.: Die zur Charakterisierung gewählten Eigenschaften für Kommunikationsprotokolle zur besseren Übersicht in einer Mindmap veranschaulicht.

2.1.1. Betrachtete Protokolle

Für den Prototypen der Verwaltungssoftware des RoRo Hafen 4.0 Projektes wird ein asynchrones Kommunikationsmodell mit Pub/Sub Pattern verwendet. Daher wird sich im Rahmen dieser Arbeit auf Protokolle beschränkt die eben dieses Muster

unterstützen und alle Protokolle ausgewählt, deren Namen in der Recherche zu dem Thema wiederholt auftauchen und am meisten Verwendung finden werden. Dabei war es egal ob Protokolle für Pub/Sub entwickelt wurden oder im Nachhinein per Erweiterung dazu befähigt wurden. Die folgenden Protokolle werden betrachtet:

Advanced Message Queuing Protocol - AMQP AMQP ist ein kostenlos nutzbares und öffentlich verfügbares Protokoll, das 2003 von John O'Hara bei JPMorgan Chase in London entwickelt wurde und dessen Version AMQP 1.0. seit Oktober 2012 als OASIS Standard anerkannt ist. Es wurde speziell für Sicherheit, Zuverlässigkeit und Interoperabilität entwickelt. Es ist ein relativ leichtes Protokoll, das dennoch viele Messagingfeatures wie Topicbasiertes Pub/Sub Routing, die Nutzung von Transaktionen und das Arbeiten mit zuverlässigen Queues bietet[Nai17].
AMQP funktioniert im Allgemeinen, indem entweder ein Publisher oder ein Consumer einen Exchange mit Namen erstellt. Dieser wird dann über den Namen für die Discovery von Publisher und Consumer verwendet[Nai17]. Auch benötigt AMQP eine Middleware die diese erstellten Exchanges und Queues verwaltet. An diesen Exchange werden dann alle betroffenen Nachrichten veröffentlicht. Der Consumer erstellt eine Queue und verbindet diese mit dem Exchange. Hierbei ist es wichtig, dass die ankommenden Nachrichten mit die Queue des Consumers gepaart werden. Dies kann auf unterschiedlichen Wegen passieren. So kann der Exchange als Fanout alle ankommende Nachrichten an alle Queues weiterleiten, oder diese direkt an einzelne Queues oder auch per Informationen im Header die Nachrichten zum richtigen Consumer routen[BPM16].
AMQP ist ein binäres Protokoll das standardmäßig TCP als Transportprotokoll verwendet und TLS/SSL und SASL benutzen kann um zusätzliche Sicherheit zu gewährleisten [Nai17]. Obergrenzen für Payload und/oder Nachrichtengröße sind anhängig davon wie der Broker/Server aufgebaut und eingerichtet ist[HKM+16]. AMQP bietet auch Stufen an Quality of Service(QoS) Einstellungen, die sich primär darin unterscheiden, wie viele Nachrichten empfangen werden dürfen bevor der Consumer sie bestätigt. Ebenso verfügt es auch über ein begrenztes kreditbasiertes Flowmanagmentsystem und Einstellungsmöglichkeiten um die Persistenz von Nachrichten zu gewährleisten[Moy15]. Beliebte Implementationen für die Middleware sind zum Beispiel ActiveMQ oder auch RabbitMQ.

Streaming Text-Oriented Message Protocol - STOMP STOMP wurde entwickelt, um Messagebroker in Skriptsprachen wie Ruby, Python oder Perl verbinden zu können. Dafür müssen üblicherweise nur eher kleine, logisch simple Aufgaben erfüllt werden. So wie das zuverlässige Senden einer Nachricht an ein Ziel oder das Abholen aller Nachrichten von einem Ziel[CI12]. Daher ist STOMP ein leichtes und

leicht nutzbares Protokoll, das einfach implementierbar sein soll. Mit der Zeit wurde STOMP durch das hinzufügen neuer Optionen langsam komplexer.
Allgemein ist STOMP ein Frame basiertes Protokoll, das HTTP nachempfunden wurde. So besteht ein Frame aus einem Befehl, optionalen Headern und optionaler Payload[CI12]. Ein Publisher muss sich erst mit dem Server verbinden und kann dann direkt Nachrichten an Queues senden. Wohingegen ein Consumer, nach dem er sich mit dem Server verbunden hat, einer Queue subscribed und dann die Nachrichten, die dort zwischengespeichert sind, erhält[CI12].
STOMP ist zwar ein textbasiertes Protokoll, erlaubt aber auch binäre Nachrichtenkörper [CI12]. Allgemein sind fast alle Einstellungen für den STOMP-Server und somit für Nachrichtendestinations, Nachrichtenattribute und Sicherheitseinstellungen vom Programmierer definierbar, da sie von den Brokereinstellungen abhängen. Somit kann STOMP sehr genau auf die eigenen Anforderungen angepasst werden, bietet dafür aber keine Unterstützung bei der Implementation ebendieser Features wie z.B. QoS Einstellungen oder Sicherheitsabfragen. Wenn man selbst keinen STOMP Server implementieren möchte, dann kann man auf die meisten größeren Messagebroker, wie ActiveMQ, RabbitMQ oder CoilMQ zurückgreifen, da diese STOMP Plugins haben und auch auf der STOMP Homepage verlinkt werden.

Message Queuing Telemetry Transport Protocol - MQTT MQTT wurde 1999 von Andy Stanford-Clark von IBM und Arlen Nipper von Arcom Control Systems Ltd (Eurotech) entwickelt[Nai17]. Es wurde als reines pub/sub Protokoll für Netzwerke mit begrenzter Bandbreite und Hardwareressourcen entwickelt. Es verwendet einen Broker, an den Nachrichten gesendet werden und der die Nachrichtenziele, hier Topic genannt, verwaltet und auch die Subscriptions zu diesen Topics[Nai17] verwaltet. Klienten können mehreren Topics subscriben und dann alle Nachrichten von dort erhalten. MQTT ist ein binäres Protokoll das mit sehr kleinen Headern arbeitet und auch eine maximale Payload von 256 MB festlegt[Nai17]. Trotzdem ist es in der Lage TLS/SSL für sichere Verbindungen zu verwenden[Moy15]. Einer der Vorteile von MQTT sind seine 3 vordefinierten Level für die Qualitiy of Service[Moy15]. Diese sind:

- Eine Nachricht wird immer nur einmal versendet.
- Die Nachricht wird mindestens einmal gesendet, es kann aber auch zu Duplikaten kommen.
- Die Nachricht wird nur genau einmal versendet; es ist aber nicht eindeutig geklärt, ob sich dieses Level wirklich implementieren lässt.

MQTT ist ein einfaches Protokoll mit wenig Kontrollmöglichkeiten für die Verwaltung und Überwachung vieler kleiner Geräte in einem Netzwerk[Nai17]. Für den Broker gibt es viele verschiedene Optionen wie Mosquitto, HiveMQ, PubSub+ oder Paho MQTT.

Kafka Kafka ist ein von der Apache Software Foundation entwickelte verteilte Plattform zum handhaben von Nachrichtenströmen[Kre17]. Es funktioniert etwas anders als die bisherigen Protokolle, da es zwar auch Topics als Ziele für Publisher verwendet, diese aber in Partitionen aufgeteilt sind, wobei eine Partition von genau einem Consumer genutzt wird, und auf die Knoten der Cluster verteilt sind um die Belastung besser verteilen zu können[Kre17]. An welche Partition eine Nachricht gesendet wird, hängt vom Producer der Nachricht ab. Der verteilte Aufbau von Kafka zeigt seine größte Stärke: seine horizontale Skalierbarkeit. Es ermöglicht auch die Echtzeiteignung des Systems. Ebenso hat Kafka die einzigartige Eigenschaft ,dass Nachrichten in Topics bis zu einem bestimmten Zeitpunkt gespeichert bleiben, auch nachdem sie von einem Consumer abgerufen wurden[Kre17]. Somit ist Kafka besonders gut für Systeme geeignet die große Anzahlen an Daten verwalten, wie etwa Logs in denen man im Fehlerfall den letzten korrekten Zustand finden will. Um dies zu ermöglichen werden Nachrichten in den Topics im Stile von Commitlogs als Record gespeichert der lediglich aus einem Tripel von Wert, Schlüssel und Zeitstempel besteht[Kre17]. Apache bietet getrennte APIs für die Teilfunktionen an und bietet Bibliotheken in vielen verschieden Programmiersprachen an, auch wenn nur die Javabibliotheken als Teil des Hauptprojektes aktiv weitergeführt werden.

Data Distribution Service - DDS DDS ist eine Netzwerkmiddleware, die entwickelt wurde, um die Nachteile von zentralisierten Publish/Subscribe Systemen zu umgehen[CK16]. Das System besteht aus einem Netzwerk aus Knoten, die sich mithilfe eines Localizationservers entweder als Publisher oder Consumer identifizieren können[CK16]. Da DDS datenorientiert ist funktioniert die Discovery über Topics, die diese Daten repräsentieren. Sobald sich zwei Knoten gefunden haben läuft die Kommunikation als peer-to-peer zwischen den Knoten und nicht mehr über den Server[CK16]. Publisher posten ihre Werte an diese Topics unabhängig davon, ob es Subscriber gibt und ebenso bei den Subscriber, die unabhängig davon ob es dort überhaupt Werte gibt bestimmten Topics subscriben . Des weiteren zeichnet DDS sich dadurch aus, das es von allen betrachteten Protokollen die meisten und komplexesten Einstellungsmöglichkeiten für QoS, Sicherheit, Durability, Zuverlässigkeit bietet[Gro15]. Die Implementierung funktioniert indem in einer Programmiersprachen-unabhängigen IDL (Interface Description Language) die Struktur eines Topics angegeben wird und dann mit einem Codegenerator daraus alle notwendigen Quelldateien generiert werden[Gro15].

Extensible Messaging and Presence Protocol - XMPP XMPP wurde ursprünglich für Instant Messaging entwickelt und wurde aus dem Jabber Protocol weiterentwickelt. XMPP basiert auf XML Streams und wurde später durch Erweiterungen angepasst um auch für publish/subscribe Anwendungen nutzbar zu sein[MSAM20].

Dafür müssen dann Nodes kreiert werden, die das Äquivalent zu Topics in anderen Protokollen sind[MSAM20]. Einen Publisher sendet dann seine Nachrichten als XML Stream an diese Node. Diese informiert dann alle, die dieser Node subscribed sind, über diese Nachrichten. Allerdings unterstützt es keine eigenen QoS Einstellungen, aber die Nutzung von TLS/SSL[MSAM20].

Simple Message Queue - SMQ SMQ ist das von Real Time Logic entwickelte Protokoll für das Internet of Things. Es wurde genau dafür entwickelt, Sensordaten von sehr vielen Geräten unter harten Echtzeitanforderungen verwalten zu können und diese Geräte zu steuern. SMQ verwendet ebenfalls einen Broker, über den die Nachrichten laufen, und ist insgesamt MQTT sehr ähnlich. Vorteile von SMQ sind seine Hardware- und Plattformagnostik, dass es Clustering unterstützt und sehr gut mit anderen Protokollen zusammenarbeitet.. Zum Beispiel kann der SMQ Broker ohne Anpassungen auf einem andern Anwendungsserver laufen[Sys16]. Auch ist es bei SMQ leichter als bei MQTT, mit einem bestimmten Endpunkt(Gerät) zu kommunizieren. Ein großer Nachteil ist, dass SMQ proprietär ist.

ZeroMQ ZeroMQ ist ein binäres Protokoll, das mit dem Fokus auf Minimalismus und der Effizienz entwickelt wurde[Hin13]. Es benutzt keinen Broker und arbeitet stattdessen mit Sockets. Es müssen also ein Pub Socket und mindestens ein Sub Socket verbunden sein, da ein Pub Socket sonst alle Nachrichten droppt[Hin13]. Dies erschwert die Discovery von neuen Sockets, da diese eigentlich per Hand verbunden werden müssen, auch wenn es Lösungen mithilfe von Proxy Sockets mit fester bekannter Adresse, die als Ankerpunkt für alle Anderen Sockets fungieren und die Nachrichten nur weiterleiten, gibt. ZeroMQ stellt die Sockets zur Verfügung, die zum Nachrichtenaustausch verwendet werden. Diese handhaben aber alle IO-Prozesse im Hintergrund, was es dem Benutzer unmöglich macht vorherzusagen wann genau Nachrichten versendet werden. Ebenso dauert der Verbindungsaufbau etwas länger, weshalb die erste gesendete Nachricht fast immer verloren geht[Hin13], was berücksichtigt werden muss. ZeroMQ liegt mehr auf der Grenze zwischen Transportprotokollen und Kommunikationsprotokollen und schickt seine Daten ähnlich zu UDP.

Java Messageing Service - JMS Der Java Messaging Service ist nicht im klassischen Sinne ein Protokoll, sondern eine Messaging Oriented Middleware(MOM) und ein Standard Java Interface, um die Funktionen und Einrichtungen von Enterprise MOM Servern nutzen zu können.[SAKB10]. Somit fungiert es als Standard zur Implementierung einer Middleware. Da es aber eine API ist, hängen die genauen Spezifikationen von den jeweiligen Implementationen ab. Ebenso löst JMS keine Sicherheitsprobleme, indem es in ein System integriert wird. Auch kann das JMS

Modell zwar skalieren, aber das liegt nur bedingt an JMS, da Systeme in denen es verwendet werden kann auch mit anderen Middlewares skalieren könnten. Dafür können mittels JMS textbasierte, binäre, stream-/ und mapbasierte Nachrichten versendet werden. Für Pub/Sub Kommunikation bietet JMS hierarchische Topics zur Verteilung an Consumer, die diesem Topic subscribed haben. Ebenso unterstützt JMS die Nutzung von Transaktionen, sowie verschiedene Modi für die Nachrichtenübertragung. So können Nachrichten persistent gesendet werden, sodass sie nicht nur im Server Memorybuffer gespeichert sind, während sie auf den Versand warten, und Subscriptions durable gewählt werden, sodass sie bei einem disconnect nicht gelöscht werden. JMS wurd für Java entwickelt und basiert auf der Java Platform Enterprise Edition. Dementsprechend ist es auch nur dafür gedacht und geeignet[Mah04].

Amazon Simple Notification Service - Amazon SNS Amazon SNS ist eine im Rahmen des Amazon AWS Ecosystems entwickelte Pub/Sub Umgebung und bietet die Möglichkeiten zur Entwicklung eines Themen-basierten pub/sub Systems. Es gibt hier eine maximale Nachrichtengröße von 256kb[Ser20] und ein Bezahlmodell, das auf der Anzahl an Anforderungen an die Amazon Server basiert, wobei eine Anfrage 64kb entspricht. Man nutzt die AWS Cloud Infrastruktur und es ist sehr viel einfacher Services, wie Email Notifications zu implementieren, als bei anderen Protokollen, da wir auch das gesamte AWS Ecosystem nutzten können. Insgesamt ließen sich aber nicht genug Vorteile gegenüber Open-Source Lösungen finden, welche die Nutzung eines solchen proprietären Systems rechtfertigen lassen. Auch war der Zugriff zu Informationen ohne AWS Account begrenzt. Das Amazon SNS wurde aus der Bewertung genommen, da der Fokus mehr auf Open- Source Lösungen liegen sollte, da diese besser untereinander vergleichbar sind und Amazon SNS im Vergleich zum ebenfalls proprietären SMQ kein Alleinstellungsmerkmal hat, das ein Einbeziehen erzwingt. Trotzdem ist dies eine Pub/Sub Umgebung, die für Unternehmen, die ohnehin bereits das AWS Ecosystem nutzen, eine Option ist.

Google Cloud Pub/Sub Google Cloud Pub/Sub ist eine von Google im Rahmen Ihrer Google Cloud entwickelte Pub/Sub Middleware. Diese bietet Themen, an die Nachrichten gesendet werden können, sowie Subscriptions an diese Themen[Goo20i]. Ebenso ist es möglich Nachrichten dauerhaft zu speichern und mithilfe der Google Cloud Echtzeitanforderungen zu erfüllen[Goo20i]. Allerdings hat sich hier eine ähnlich Situation wie bei Amazons SNS ergeben, weshalb Google Cloud Pub/Sub aus den selben Gründen aus der Bewertung genommen wurde. Ebenso wie Amazon SNS ist dies eine hervorragende Option für Projekte die ohnehin schon Google Cloud nutzten. Auch bei Google Cloud Pub/Sub basiert das Monetarisierungsmodell auf dem Umfang der Zugriffe auf die Google Cloud.

2.2. Logistikdomäne Osteeraum

Im Rahmen dieser Arbeit wird eine sehr vereinfachten Definition der Logistikdomäne Ostseeraum, in der sie sich in 2 Komponenten aufteilt, verwendet. Diese Komponenten sind die Häfen des Ostseeraums und die Schiffe, die diesen befahren. Von diesen werden auch die Events erzeugt, die als Nachrichten versendet werden müssen. Mithilfe dieser Events wird der Verwaltungssoftware ermöglicht ihren Job zu machen und diese Schiffe und Häfen zu verwalten. Im Ostseeraum gibt es ca. 200 Logistikhäfen, wobei die Top 20 etwa 66% des Gesamtumschlages ausmachen und die Top 3 sogar etwa 25%. Für den RoRo-Verkehr(Roll on Roll off -Verkehr) stellen die Top 20 sogar ca. 82% des Umschlags dar und beim Container-Verkehr 95%. An allen Häfen im Ostseeraum gab es im Jahr 2015 etwa 295.000 Besuche in Häfen. Davon waren 46% Passagierschiffe, ca. 14% General Cargo, ca. 13% Tanker und jeweils ca. 5% RoRo- und Containerschiffe. Insgesamt operierten in 2015 etwa 7900 Schiffe im Ostseeraum und es waren zu jedem Zeitpunkt ca. 1500 IMO-Registrierte Schiffe gleichzeitig im Ostseeraum aktiv. Diese Schiffe sind durch ihre IMO Nummer identifizierbar. 1987 wurde von der Internationalen Seeschifffahrts-Organisation diese Kennzeichnung als ein Standard zur Identifizierung von Schiffen eingeführt[Org88]. Seit 1996 sind diese Nummer für Schiffe verpflichtend. Es müssen IMO-Nummern an Frachtschiffe mit einer Bruttoraumzahl oder Gross-Tonnage von mindestens 300 sichtbar sein und ebenso an Passagierschiffen mit einer Bruttoraumzahl von mindestens 100. Wobei zu beachten ist das alle Schiffe Passagierschiffe sind, die keine Frachtschiffe sind. Ausgenommen davon sind:[Org20]

- Fischereifahrzeuge
- Schiffe ohne mechanischen Antrieb
- Freizeit-Yachten
- Spezialschiffe, z. B. Feuerschiffe, SAR-Schiffe
- Schuten
- Tragflächenboote, Luftkissenfahrzeuge
- Schwimmdocks
- Kriegsschiffe
- Schiffe aus Holz

Von den etwa 7900 Schiffen im Ostseeraum waren 48% General-Cargo-Schiffe, 22% Tanker, 5,4% Passagierschiffe und 3,1% RoRo-Cargo-Schiffe.

2.2.1. Eventquellen

Häfen lösen Events aus, bei der Abfertigung von Schiffen und deren Transportgütern. Im Lübecker Hafen produziert jede Einheit, Container oder Trailer, die auf

dem Schiff transportiert wird, bei einem Hafenbesuch im Schnitt 3-4 Events. Für Eintreffen, Umpositionierung auf dem Gelände und das Verlassen. Ebenso löst jedes Schiff und jeder Zug auch im Schnitt 4 Events aus (Eingang, Anfang/Ende Abfertigung, Ausgang). Schiffe produzieren im Vergleich dazu auch außerhalb eines Hafens Events, da sie alle 2 Minuten Ihre Position, Heading und Geschwindigkeit mitteilen.

2.2.2. Eventstruktur

Zurzeit ist für Events noch keine bestimmte Struktur festgelegt, weshalb diese Arbeit sich darauf beschränkt ihre Größe abzuschätzen. Dies geschieht über Ihre bekannten gemeinsamen Elemente. Die Kernelemente werden alle Events haben müssen:

- ID: Die Universally Unique Identifier(UUID) des Events und somit eine 128 bit Zahl (128bit = 16B).
- Quelle: Ein eindeutiger Name des Event-Autors. Optionen dafür wären IMO-Nummer(10 Char), BICLOCODE(9 Char)oder MRN (15+ Char).
- Zeitstempel (nach ISO 8601): Dieser besteht aus 25 Charakteren.
- Typ: Eine eindeutige Identifizierung der Art des Events(15 Char).

Zu diesen Abschätzungen kommen noch die Buchstaben für die Meta- Strukturinformationen des Events selbst (im Falle JSON beispielsweise die Klammern, Kommata und Attributsnamen). Im Schnitt wird davon ausgegangen, dass etwa 12 Buchstaben pro Attribut dazu kommen. Zu den oben aufgeführten Kernattributen werden Events noch Typ-Spezifische Informationen beinhalten. Die Anzahl dieser Attribute wird stark zwischen Event-Typen variieren. Für den Rahmen dieser Arbeit wird die Anzahl dieser Attribute auf im Regelfall 10 festgelegt.
Daraus ergibt sich ein Richtwert von 215 Buchstaben pro Event. Da wir von einer UTF-8 Codierung ausgehen ergibt sich eine Größe von etwa 450B pro Event. Dies ist auch der Richtwert für die Datenmenge die in einer Nachricht versendbar sein soll. Dies entspricht dem Richtwert für die meisten erwarteten Events. Da es aber auch kleine Events ohne Attribute geben wird, kann die Größe der Events hier auf 220B geschätzt werden und ist somit die minimale Datenmenge die in einer Nachricht versendbar sein muss. Es kann ebenso gut auch Events geben, die deutlich mehr als diese 10 Attribute beinhalten und da es nach oben keine Grenzen gibt, solange die Attribute sich an die Strukturvorgaben halten, wird im Rahmen dieser Arbeit und zu Testzwecken die Anzahl der Attribute solcher Events auf 20 festgelegt.

3. Konzept

In diesem Kapitel wird erläutert, wie die Kriterien und Bewertungsmaßstäbe entstanden sind und wie daraus die Vergleichsanalyse entwickelt wurde. Im Anschluss werden die Ergebnisse aus der Vergleichsanalyse dargestellt und es wird definiert wie diese Ergebnisse im Rahmen einer praktischen Implementierung geprüft und getestet wurden.

3.1. Betrachtete Attribute für theoretische Analyse

Um die in Kapitel 2 gewählten Charaktereigenschaften für Protokolle besser bewerten und analysieren zu können, wurden auf Basis von ihnen Kategorien entwickelt. Diesen wurden dann wiederum Attribute zugewiesen, die diese besser beschreiben. Diese Kombination aus Kategorien und Attributen ermöglicht die Entwicklung einer nachvollziehbaren Bewertungsskala für Kommunikationsprotokolle, die auf einer genaueren Bewertung jeder einzelnen Kategorie basiert. Folgende Kategorien wurden gewählt:

Nachrichten Diese Kategorie beschreibt primär, wie sich das Protokoll und seine Nachrichtenstruktur auf die mögliche Bandbreite und Art der Übertragung auswirkt.

- Headergröße: Dies ist der wichtigster Faktor für den Overhead des Protokolls und beeinflusst maßgeblich wie groß der Datenverkehr des Protokolls im Schnitt wird und hat somit auch großen Einfluss auf die mögliche Anzahl an Nachrichten über eine bestimmte begrenzte Verbindung.
- Kodierung: Es wird für die Events von einer UTF-8 Kodierung von Strings ausgegangen. Somit erleichtern binäre oder stringbasierte Protokolle die Arbeit der Software da hier, im Gegensatz zu anderen Datenformaten, keine weiteren oder nur marginale Kodierungsvorgänge notwendig sind. Zusätzlich ist für solche Vorgänge Rechenleistung erforderlich die potentiell ressourcenbeschränkte Clienten zusätzlich belastet und dadurch deren Arbeitsgeschwindigkeit beeinflussen kann.

- max.Payload: Da es Protokolle mit starken Beschränkungen gibt, ist es relevant wo diese Grenzen liegen und ob diese das Versenden mehrerer Nachrichten bedingen, da mehrere Nachrichten zu mehr Komplexität und Overhead führen und bei begrenzten Verbindungen die maximal mögliche Nachrichtenanzahl beschränken.

Sicherheit Für alle Netzwerke ist Sicherheit ein relevanter Faktor und je größer und komplexer das Netzwerk, desto wichtiger wird es verschiedene Optionen für verschieden Verbindungen zu haben, um trotz Erfüllung von Sicherheitsvorgaben eine gewisse Geschwindigkeit in der Nachrichtenübertragung zu gewährleisten.

- Optionen: Hier liegt der Fokus auf den im Protokoll enthaltenen Optionen zur Gewährleistung sicherer Nachrichtenübertragung.

Skalierbarkeit Um ein später wachsendes Netzwerk verwalten zu können müssen Protokolle und Ihre Implementierungen in der Lage sein zu skalieren und diese Eigenschaft wird in dieser Kategorie abgebildet.

- Optionen: Fast alle Protokolle können vertikal skalieren, mit mehr Rechenleistung. Es ist aber relevant zu wissen ob Protokolle Grenzen bei diese Skalierung haben oder ob sie nicht sogar horizontal skalieren können. Gerade für große Anwendungen ist dies eine wichtige Option da wir somit mit auch mit mehren Knoten auf eine steigende Belastung in der Zukunft reagiert werden kann.

Hardwarebelastung Gerade wenn Protokolle im IoT verwendet werden sollen ist einer ihrer wichtigsten Eignungsfaktoren die von Ihnen produzierte Hardwarebelastung. Wobei die Belastung für Klienten wichtiger ist, da diese häufiger ressourcenbeschränkt sind als die Broker. Aber auch auf deren Seite ist das Ausmaß der benötigten Ressourcen für die Planung wichtig.

- Belastung Broker: Wie groß das Rechenzentrum für die Middleware sein muss oder ob bereits vorhandene PCs genügen, ist ein wichtiger Faktor bei der Bewertung eines Protokolls.
- Belastung Client: Gerade um eine Iot-Fähigkeit zu ermöglichen ist es, wichtig die Belastung für die Klienten so gering wie möglich zu halten, um auch hier eventuell bestehende Hardware mitnutzen zu können oder nur möglichst kleine leichte, kostensparende Hardware anschaffen zu müssen.

Middleware In dieser Kategorie wird definiert ob ein Protokoll eine Middleware benötigt und, wenn ja, wie einfach es ist diese zu nutzen und aufzusetzen.

- benötigt: Da die Middleware eine Entkopplung der Teilnehmer darstellt kann sie in machen Anwendungsfällen alleine dafür wichtig sein.
- Anzahl Optionen: Hierbei geht es vor allem darum wie viele Möglichkeiten es gibt, die Middleware zu implementieren bzw. wie viele Implementationen vorhanden sind, auf die zurückgegriffen werden kann.
- Anzahl Sprachen: Für die bereits vorhanden Optionen ist noch entscheidend für welche Programmiersprachen bereits eine Bibliothek zur Implementierung entwickelt wurde, um möglichst vielen Menschen eine einfache Nutzung des Protokolls zu ermöglichen.

Quality of Service Für alle Protokolle in allen Anwendung ist Zuverlässigkeit ein wichtiger Einfluss. Stark beeinflussen lässt sich diese von den einzelnen Protokollen durch zusätzliche Einstellungsoptionen.

- Optionen: Ob es diese Optionen gibt und, wenn ja, wie genau diese aussehen, wird in diesem Attribut bewertet. Dabei ist es wie in fast allen Bereichen auch hier besser wenn das Protokoll dem Nutzer mehr Optionen zur Verfügung stellt.

Ersteinstieg & Support Um die Langzeitnutzbarkeit und Schwierigkeit bei der Einarbeitung in ein Protokoll bewerten zu können wurde diese Kategorie mit gewählt. Diese legt einen recht großen Fokus auf die Community rund um dieses Protokoll, da diese viel Hilfe, Unterstützung und Weiterentwicklung liefert und auch ein Indiz bietet wie lange ein Protokoll noch weiterentwickelt und am Leben gehalten wird.

- exist. Libraries: Hierbei wird dargestellt wie viele bereits nutzbare Bibliotheken zum implementieren von Klienten es für das Protokoll bereits gibt.
- proprietär: Diese Eigenschaft stellt eine Schranke da und macht den Nutzer sehr abhängig vom Herausgeber des Protokolls, was Weiterentwicklung, Langzeitunterstützung und teilweise Support betrifft.
- Open Source: Als Gegenpart zur Proprieärität hilft uns Open Source Entwicklung da es leichter eine große Community ermöglicht und somit die Einarbeitung in das Protokoll und eine Weiterentwicklung vereinfacht.
- letzter Github Commit: Da die meisten, gerade Community getriebenen, Softwareentwicklungen auf Github veröffentlicht sind, wurde dieses Attribut als Indiz gewählt um Einschätzen zu können wie lebendig eine Community ist und wie aktiv die Entwicklung noch voran getrieben wird.

- Google Trends: Dieses Attribut ist als zweites Indiz gewählt um einzuschätzen wie beliebt und viel genutzt dieses Protokoll ist und somit wie hoch die Chancen sind auch in der Zukunft noch eine große Community vorzufinden.

Adressierung Diese Kategorie umfasst die Optionen des Protokolls bei der Zustellung von Nachrichten. Wie können Empfänger adressiert werden, wer initiiert die Nachrichtenübertragung. Außerdem auch ob Nachrichten auf dem Broker, meistens in Queues, persistent zwischen gespeichert werden.

- Topic hierarchisch: Die Eigenschaft hierarchischer Topics ermöglicht eine leichtere Adressierung und bietet Vorteile bei der Verwaltung vieler Nachrichtenziele, da eine solche Strukturierung sehr intuitiv funktioniert.
- Queues (persistent): Da es zu Ausfällen von Klienten und auch Brokern kommen kann ist wichtig über ein Protokoll einschätzen zu können wie es mit noch nicht zugestellten Nachrichten in einem solchen Fall umgeht und ob sie zwischengespeichert werden oder verloren gehen
- Routing per Topic: Topics erlauben leichteres und komplexeres Routing von Nachrichten an verschiedene Empfänger. Die Information über die Existenz dieser Option ist wichtig um ein Protokoll vollständig zu beschreiben.
- Zustellungsrichtung: Um feststellen zu können, für welche Anwendungsfälle ein sich eignet Protokoll, ist es wichtig zu wissen ob ein Klient sich Nachrichten vom Server holen muss oder ob dieser Nachrichten bei auftreten an die Klienten pusht, da sich dadurch die benötigte Softwarearchitektur verändern kann.

Discovery Gerade in Netzwerken mit vielen Teilnehmern, von denen einige das Netzwerk immer wieder verlassen oder neu betreten, ist das Handling des Entdeckens der Teilnehmer ein wichtiger Faktor.

- Optionen: Hierbei wird wiedergegeben ob ein Protokoll dynamisch neue Teilnehmer finden und verbinden kann, und wenn dies nicht der Fall ist, wie stark der Nutzer an diesem Prozess beteiligt werden muss.

Transport Spiegelt wieder, wie der Nachrichtentransport mit einem Protokoll funktioniert. Primär, welche Zeitanforderungen dieses Protokoll erfüllen kann, da dies einen maßgeblichen Einfluss hat welches Transportprotokoll verwendet wird.

- unterstützte Zeitanforderungen: Hierbei wird betrachtet wie gut ein Protokoll für Echtzeitanforderungen geeignet ist und ob es sogar für harte Echtzeitanforderungen geeignet ist.

- verwendetes Transportprotokoll: Da das unterstützte Transportprotokoll die Sicherheit und Qualität der Übertragung mit beeinflusst, ist es für unsere Einschätzung des Protokolls relevant.

3.2. Bewertungsmaßstab der theoretischen Analyse

Diese festgelegten Kategorien sind möglichst allgemein gehalten um mit anderer Gewichtung auch für andere Anwendungsfälle nutzbar zu sein. Um aber diese für eine Bewertung anhand der Logistikdomäne Ostseeraum nutzen zu können muss noch festgelegt werden, welche Werte unsere Attribute annehmen können und wie diese gewichtet sind. Der Anwendungsfall dieser Arbeit legt seinen Fokus auf geringe Hardwarebelastung um eine IoT Nutzbarkeit zu ermöglichen, eine gute Skalierung und eine gute Entkoppelung von Sendern und Empfängern. Ebenso ist es wichtig, dass viele Eventnachrichten strukturiert verwaltet werden können, sei es nach Absender oder Wichtigkeit oder Eventtyp. Daher wurden die Optionen zur Adressierung höher gewichtet, wohingegen die Zeitanforderungen für uns nicht relevant sind. Ein weiterer Fokus, auf Communitysupport und umfangreichen Einarbeitungshilfen, geht aus der Definition des Anwendungsfalles hervor. Da sich eine mögliche Umsetzung des Anwendungsfalles noch am Anfang steht, wo viele Optionen und Hilfen wichtig sind, da noch nicht bekannt ist wer an dem Projekt arbeiten wird. Auch dauert es noch bis die Software aktiv genutzt wird, weshalb es wichtig ist ,dass das gewählte Protokoll in der Zukunft noch aktuell ist und von Entwicklern und Community unterstützt wird.
Da es sich aber als zu grob herausgestellt hat, wenn die Kategorien als ganze gewichtet werden, wurde alle Attribute eine eigene Gewichtung zugeschrieben. Auch wurden aufgrund der Ungewissheit, bezüglich des Projektes, mehr Optionen als besser zu bewertet, um die Entwicklung möglichst wenig einzuschränken. Des weiteren wurde versucht für alle Attribute 3 mögliche Werte zu definieren damit es pro Attribut 0, 1 oder 2 Punkte mal Gewichtung geben kann. Dies ist größtenteils gelungen. Aus den verteilten Punkten ergibt sich eine Punktzahl, die als Prozentwert von den Gesamtpunkten dargestellt wird. Auf Basis dieser Überlegungen wurde eine Bewertungsmatrix erstellt die in Tabelle 3.1 dargestellt ist.

Protokoll			Attribut/Punkte		Attribut/Punkte		Attribut/Punkte	
Nachrichten	Headergröße	2x	groß	0	mittel	1	klein	2
	Kodierung	2x	other	0	text	1	binär	2
	max. Payload	3x	<450 byte	0	>= 450 byte	1		
Sicherheit	Optionen	3x	best Practice	0	Netzstandard	1	eigene Standards	2
Skalierbarkeit	Optionen	3x	gar nicht	0	vertikal	1	horizontal	2
Hardwarebelastung	Belastung Broker	2x	hoch	0	mittel	1	gering	2
	Belastung Client	3x	hoch	0	mittel	1	gering	2
Middleware	benötigt	3x	nein	0	ja	1		
	Anz. Optionen	2x	1	0	2-5	1	>5	2
	Anz. Sprachen	2x	1	0	2-5	1	>5	2
Quality of Service	Optionen	2x	best Practice	0	vordef. Level	1	individ. Einst.	2
Ersteinstieg & Support	exist. Libraries	2x	1	0	2-5	1	>5	2
	proprietär	3x	ja	0	nein	1		
	Open Source	1x	nein	0	ja	1		
	lzt. Github Commit	3x	last year	0	last Month	1	last week	2
	GoogleTrends	2x	<Kafka	0	=Kafka	1	>Kafka	2
Addressierung	Topic hierachisch	2x	nein	0	ja	1		
	Queues (persis.)	2x	nein	0	eine	1	mehrere	2
	Routing per Topic	3x	nein	0	ja	1		
	Zustellungsricht.	1x	Client pullt	0	Server pusht	1	beides	2
Discovery	Optionen	2x	manuell	0	statisch	1	dynamisch	2
Transport	erfüllte Zeitanf.	1x	verzögert	0	quasi Echtzeit	1	Echtzeit	2
	Transportprot.	2x	eigenes	0	UDP	1	TCP	2

Tabelle 3.1.: Bewertungsmatrix für die Kommunikationsprotokolle

3.3. Ergebnisse theoretische Analyse

Auf Basis der Recherche zu den betrachteten Protokollen und nach Anwendung der Bewertungsmatrix ergibt sich eine Reihenfolge der Protokolle die in den Tabellen 3.2 bis 3.6. dargestellt ist. Zusätzlich geben die Tabellen einen guten Überblick über die Eigenschaften der Protokolle.

Protokoll			AMQP		Stomp	
			Inhalt	Pkt	Inhalt	Pkt
Nachrichten	Headergr.	2x	mittel	2	klein	4
	Kodierung	2x	binär	4	binär	4
	max. Payload	3x	>=450 b	3	>=450b	3
Sicherheit	Optionen	3x	eig. Stand.	6	best Pract.	0
Skalierbarkeit	Optionen	3x	vertikal	3	vertikal	3
Hardwarebel.	Bel. Broker	2x	hoch	0	gering	4
	Bel. Client	3x	hoch	0	gering	6
Middleware	benötigt	3x	ja	3	ja	3
	Anz. Optionen	2x	>5	4	>5	4
	Anz. Sprachen	2x	>5	4	>5	4
QoS	Optionen	2x	vordef. Level	2	best Pract.	0
Einst.&Supp.	exist. Libraries	2x	>5	4	>5	4
	proprietär	3x	nein	3	nein	3
	Open Source	1x	ja	1	ja	1
	lzt. Githubcommit	3x	last week	6	last Month	3
	GoogleTrends	2x	<Kafka	0	<Kafka	0
Addressierung	Topic hierachisch	2x	ja	2	nein	0
	Queues(persis.)	2x	mehrere	4	mehrere	4
	Routing per Topic	3x	ja	3	ja	3
	Zustellungsricht.	1x	beides	2	beides	2
Discovery	Optionen	2x	dynamisch	4	statisch	2
Transport	erfüllte Zeitanf.	1x	quasi Echtzeit	1	Echtzeit	2
	Transportprot.	2x	TCP	4	TCP	4
Summe	Punkte		65/87		63/87	
	Prozent von Max		74,71 %		72,41 %	

Tabelle 3.2.: Die Ergebnisse der Vergleichsanalyse für die Protokolle AMQP, STOMP in der Reihenfolge der erreichten Punkte dargestellt.

Protokoll			MQTT		Kafka	
			Inhalt	Pkt	Inhalt	Pkt
Nachrichten	Headergr.	2x	klein	4	klein	4
	Kodierung	2x	binär	4	text	2
	max. Payload	3x	>=450b	3	>=450b	3
Sicherheit	Optionen	3x	best Pract.	0	eigene Stand.	6
Skalierbarkeit	Optionen	3x	vertikal	3	horizontal	6
Hardwarebel.	Bel. Broker	2x	gering	4	hoch	0
	Bel. Client	3x	gering	6	hoch	0
Middleware	benötigt	3x	ja	3	ja	3
	Anz. Optionen	2x	>5	4	1	2
	Anz. Sprachen	2x	>5	4	>5	4
QoS	Optionen	2x	vordef. Level	2	vordef. Level	2
Einst.&Supp.	exist. Libraries	2x	>5	4	>5	4
	proprietär	3x	nein	3	nein	3
	Open Source	1x	ja	1	ja	1
	lzt. Githubcommit	3x	last week	0	last week	6
	GoogleTrends	2x	>Kafka	4	=Kafka	2
Addressierung	Topic hierachisch	2x	ja	2	nein	0
	Queues(persis.)	2x	eine	2	mehrere	4
	Routing per Topic	3x	ja	3	ja	3
	Zustellungsricht.	1x	Server pusht	1	Client pullt	0
Discovery	Optionen	2x	manuell	0	statisch	2
Transport	erfüllte Zeitanf.	1x	quasi Echtzeit	1	quasi Echtzeit	1
	Transportprot.	2x	TCP	4	TCP	4
Summe	Punkte		62/87		62/87	
	Prozent von Max		71,26 %		71,26 %	

Tabelle 3.3.: Die Ergebnisse der Vergleichsanalyse für die Protokolle MQTT, Kafka in der Reihenfolge der erreichten Punkte dargestellt.

Protokoll			DDS		XMPP	
			Inhalt	Pkt	Inhalt	Pkt
Nachrichten	Headergr.	2x	klein	4	mittel	2
	Kodierung	2x	other	0	other	0
	max. Payload	3x	>= 450b	3	>=450b	3
Sicherheit	Optionen	3x	eigene Stand.	6	Netzstand.	3
Skalierbarkeit	Optionen	3x	vertikal	3	vertikal	3
Hardwarebel.	Bel. Broker	2x	mittel	2	gering	4
	Bel. Client	3x	gering	6	gering	6
Middleware	benötigt	3x	nein	0	ja	3
	Anz. Optionen	2x	>5	4	>5	4
	Anz. Sprachen	2x	2-5	2	>5	4
QoS	Optionen	2x	Individ. Einst.	4	best Pract.	0
Einst.&Supp.	exist. Libraries	2x	1	0	>5	4
	proprietär	3x	nein	3	nein	3
	Open Source	1x	ja	1	ja	1
	lzt. Githubcommit	3x	last year	6	last year	0
	GoogleTrends	2x	<Kafka	0	<Kafka	0
Addressierung	Topic hierachisch	2x	nein	0	nein	0
	Queues(persis.)	2x	ja	2	ja	2
	Routing per Topic	3x	ja	3	ja	3
	Zustellungsricht.	1x	Server pusht	1	Server pusht	1
Discovery	Optionen	2x	dynamisch	4	dynamisch	4
Transport	erfüllte Zeitanf.	1x	Echtzeit	2	quasi Echtzeit	1
	Transportprot.	2x	TCP	2	TCP	4
Summe	Punkte		58/87		55/87	
	Prozent von Max		66,66 %		63,21 %	

Tabelle 3.4.: Die Ergebnisse der Vergleichsanalyse für die Protokolle DDS, XMPP in der Reihenfolge der erreichten Punkte dargestellt.

Protokoll			SMQ		ZeroMQ	
			Inhalt	Pkt	Inhalt	Pkt
Nachrichten	Headergr.	2x	klein	4	klein	4
	Kodierung	2x	binär	4	binär	4
	max. Payload	3x	3 byte	0	<450b	0
Sicherheit	Optionen	3x	Netzstand.	3	eigene Stand.	6
Skalierbarkeit	Optionen	3x	vertikal	3	horizontal	6
Hardwarebel.	Bel. Broker	2x	gering	6	gering	4
	Bel. Client	3x	gering	4	gering	6
Middleware	benötigt	3x	ja	3	nein	0
	Anz. Optionen	2x	2-5	2	0	0
	Anz. Sprachen	2x	>5	4	1	0
QoS	Optionen	2x	best Pract.	0	best Pract.	0
Einst.&Supp.	exist. Libraries	2x	2-5	2	>5	4
	proprietär	3x	ja	0	nein	3
	Open Source	1x	ja	1	ja	1
	lzt. Githubcommit	3x	last year	0	last week	6
	GoogleTrends	2x	<Kafka	0	<Kafka	0
Addressierung	Topic hierachisch	2x	ja	2	nein	0
	Queues(persis.)	2x	nein	0	nein	0
	Routing per Topic	3x	ja	3	ja	3
	Zustellungsricht.	1x	Server pusht	1	Server pusht	1
Discovery	Optionen	2x	dynamisch	4	statisch	2
Transport	erfüllte Zeitanf.	1x	Echtzeit	2	Echtzeit	2
	Transportprot.	2x	TCP	4	eigenes	0
Summe	Punkte		52/87		52/87	
	Prozent von Max		59,77 %		59,77 %	

Tabelle 3.5.: Die Ergebnisse der Vergleichsanalyse für die Protokolle SMQ, ZeroMQ in der Reihenfolge der erreichten Punkte dargestellt.

Protokoll			JMS	
			Inhalt	Pkt
Nachrichten	Headergr.	2x	groß	0
	Kodierung	2x	text	2
	max. Payload	3x	>=450b	3
Sicherheit	Optionen	3x	Netzstand.	3
Skalierbarkeit	Optionen	3x	vertikal	3
Hardwarebel.	Bel. Broker	2x	mittel	2
	Bel. Client	3x	mittel	3
Middleware	benötigt	3x	ja	3
	Anz. Optionen	2x	>5	4
	Anz. Sprachen	2x	2-5	2
QoS	Optionen	2x	best Pract.	0
Einst.&Supp.	exist. Libraries	2x	2-5	2
	proprietär	3x	nein	3
	Open Source	1x	ja	1
	lzt. Githubcommit	3x	/	0
	GoogleTrends	2x	<Kafka	0
Addressierung	Topic hierachisch	2x	ja	2
	Queues(persis.)	2x	ja	2
	Routing per Topic	3x	ja	3
	Zustellungsricht.	1x	Server pusht	1
Discovery	Optionen	2x	statisch	2
Transport	erfüllte Zeitanf.	1x	verzögert	0
	Transportprot.	2x	TCP	4
Summe	Punkte		45/87	
	Prozent von Max		51,72 %	

Tabelle 3.6.: Die Ergebnisse der Vergleichsanalyse für das Protokoll JMS.

Auf Basis der Vergleichsanalyse wird angenommen das AMQP am besten für unseren Anwendungsfall geeignet ist. Da dies aber nur auf theoretischen Informationen basiert und sich innerhalb der Implementierung noch Probleme oder Vorteile eines Protokolls herausstellen können, wird diese Aussage im Rahmen dieser Arbeit durch ein Praxistest in Form einer Testimplementation überprüft. Da die ersten 4 Plätze alle recht dicht beieinander sind, in der theoretischen Bewertung, wäre es zu bevorzugen einen Test für alle 4 durchzuführen. Dies hätte allerdings den Zeitrahmen dieser Arbeit gesprengt. Deshalb wurde sich dazu entschlossen die Annahme zu überprüfen, indem das beste Protokoll mit der besten Alternative verglichen wird. Es wurde sich daher entschlossen eine Testimplementation von AMQP und STOMP

durchzuführen, da die Wahrscheinlichkeit die beste Option zu finden, unter den bestbewerteten Protokollen am höchsten ist.

3.4. Praktische Implementierung

Dieser Abschnitt beschreibt wie die Bedingungen für die Testimplementation gewählt wurden, nach welcher Zielsetzung die Umsetzung passierte und welche Tests auf die Implementierung angewendet wurden.

3.4.1. Broker

Für die Implementierung ist die erste elementare Frage welcher Broker verwendet wird. Kandidaten für AMQP wären ActiveMQ, RabbitMQ, Solace PubSub+, SwiftMQ und für STOMP ActiveMQ, RabbitMQ, CoilMQ. Diese haben es in die Auswahl geschafft, da sie die aktuell meistverwendeten und beliebtesten sind oder von den Protokollentwicklern als Option empfohlen werden. Auch wurde sich für eine Auswahl der beliebtesten Varianten entschieden, da man hier eine große, lebendige Community finden werden, welche Einstieg, Hilfesuche und Langzeitunterstützung erleichtert. Allerdings besteht die Gefahr bei einer Implementierung, dass am Ende eher die Implementierung im speziellen als das Protokoll im allgemeinen getestet wird. Dies soll möglichst vermieden werden. Einer der wichtigsten Faktoren darin ist die gewählte Middleware. Um diesen Faktor zu minimieren wurde sich auf Optionen beschränkt, die beide Protokolle unterstützen. Dies sind ActiveMQ und RabbitMQ. ActiveMQ wird etwas häufiger verwendet und basiert auf Java, was dem Autor der Arbeit entgegenkommt. Außerdem wird AMQP bis Version 1.0. unterstützt und gerade für STOMP wird es eher empfohlen. Daher fiel die Wahl zuerst auf ActiveMQ. Da sich allerdings die Dokumentation der API als umständlich und die gegebenen Hilfestellungen als unzureichend herausgestellt haben, wurde RabbitMQ noch einmal untersucht. Und obwohl RabbitMQ nativ AMQP nur bis Version 0.9.1. unterstützt, kann es aber auch mit Version 1.0. umgehen. Durch Plugins sind sowohl natives STOMP als auch STOMP über Websocket unterstützt. Da dort eine deutlich einsteigerfreundliche Dokumentation vorzufinden war konnte sehr viel schneller, als zuvor bei ActiveMQ, Fortschritt verzeichnet werden. Deshalb wurde am Ende RabbitMQ für die Tests zu verwendet.

3.4.2. AQMP

Um eine möglichst hohe Kompatibilität mit ActiveMQ zu garantieren wurde sich für die ebenfalls von Apache entwickelte Java Bibliothek Apache Qpid Proton entschieden. Hier wurden auch die meisten Probleme in der Dokumentation festgestellt, da es sich als sehr schwierig herausgestellt hat einen Einstieg zu finden. Nach dem Wechsel des Broker wurde der selbe Weg gegangen und RabbitMQs eigene Java Bibliothek für AMQP gewählt. Zusammen mit der guten Dokumentation war es dort sehr leicht ein einfaches, funktionierendes System aufzubauen. Allgemein wurden 2 Typen von Klienten erstellt. Einen um Nachrichten zu senden und den anderen zum Empfangen und zur Ausgabe auf dem Terminal. Da ein Praxistest mit einem Sender und einem Empfänger für den angepeilten Nutzraum wertlos ist, wurde sich dazu entschieden mehrere Klienten für den Broker als reine Verbindungen darzustellen, da es für den Broker keinen Unterschied machen sollte ob die Verbindungen alle vom selben Programm kommen oder nicht. Dadurch ließ sich umsetzten das sowohl Broker als auch alle Klienten auf einem PC laufen können, sodass ein Publisher eine bestimmte Zahl an Verbindungen zum Broker aufbaut und dann in einer Schleife über alle eine Nachricht sendet. Dies ist zwar nicht komplett synchron, geht aber schnell genug ,dass es keine Probleme darstellen dürfte. Dies wurde in den Tests bestätigt siehe Tabelle 4.3 und Tabelle 4.4 bei der Versandzeit. Um alle Nachrichtenoptionen aus Unterabschnitt 2.2.2 abbilden zu können wurden 3 String Bodys gebaut die der Eventstruktur entsprechen. Insgesamt werden die Events im Ganzen im Körper der Nachricht versendet und nicht die Header des Events auf die Header der Nachricht gematched, da dies den Vorteil hat ,dass die Kommunikation zwischen mehren Netzwerken leichter ist, gerade falls unterschiedliche Protokolle interagieren. Ebenso sind die Routingoptionen die durch das Headermatchen enstehen für die Tests nicht essentiell, da sich ihre Funktionalität trotzdem testen lässt, da der primäre Unterschied darin besteht, woher der Routingkey bezogen wird. Während der Publisher nur Zeitstempel ausgibt für Programmstart und Anfang/Ende der Sendeschleife so gibt unser Consumer für jede Nachricht den Zeitstempel und die ID aus, sodass nachvollziehbar ist ob alle Nachrichten empfangen wurden.

3.4.3. STOMP

Bei STOMP wurde nicht versucht einen ActiveMQ Clienten zu bauen, da der Versuch vorher abgebrochen wurde. Bei RabbitMQ wurde versucht auch in Java zu implementieren um eventuelle Einflüsse durch die Programmiersprache zu eliminieren. Daher wurden die beiden verfügbaren Bibliotheken Stampy oder Stilts betrachtet. Beide sind aber von Ihrer Dokumentation mehr für Anwender geeignet, die bereits Erfahrung mit der Implementierung von STOMP haben. Da es nicht möglich war eine zufriedenstellende Lösungen entwickelt werden konnte, wurde auf

Stomp.py gewechselt, da STOMP für Skriptsprachen gemacht wurde und Python deutlich einsteigerfreundlicher, als Javascript, ist. Denn zu STOMP ließen sich primär Hilfen und Beiträge zur Javaascript Variante STOMP.js oder STOMP über Websocket mittels Spring finden sowie Stomp.py. Stomp.js war keine Option, da es notwendig gewesen wäre sich komplett neu in Javascript einzuarbeiten. Ebenso sollte natives STOMP und nicht STOMP über Websocket getestet werden, was die Spring-Optionen wegfallen ließ und die Entscheidung zu Stomp.py bedingte. In der Implementierung wurde genauso wie bei AMQP vorgegangen, um die beiden Implementationen so ähnlich wie möglich zu halten. Ein einziger Unterschied ist, dass bei Stomp.py die Nachrichtenausgabe und potentielle Bestätigung über einen Listener auf der Verbindung funktioniert, der ausgelöst wird wann immer eine Nachricht über diese Verbindung registriert wird.

3.4.4. Tests

Um definieren zu können welche Test durchführt werden müssen wurde festgelegt, welche Anwendungsfälle der Broker später bewältigen muss. Das sind einmal das Positionslog der Schiffe. Also alle 2 Minuten bis zu 8000 Nachrichten die innerhalb dieser 2 min korrekt gesendet und empfangen werden müssen, mit wenigen bis keinen Fehlern bzw. Verlusten. Und im zweiten Fall die konstant ausgelösten Events im Hafen, durch ankommende und bearbeitete Schiffe und ihre Last. Da sich hier über die Ankunftszeit der Events oder das Muster beim Auslösen keine Aussagen treffen lassen, fiel die Entscheidung diesen Fall durch einen Bandbreitentests über einen bestimmten Zeitraum zu testen. Dabei wurde überprüft ob die gefunden Bandbreite für die zu erwartende Belastung ausreicht. Alle Tests wurden auf einem PC mit Windows 10 64-bit, 4GB RAM und einem AMD Athlon II X4 640 Prozessor mit 3 Ghz durchgeführt, wobei ein Laptop mit Windows 10 64-bit, 4 GB RAM und einem Intel Pentium N3710 CPU mit 2x 1,60 Ghz als zweiter Knoten für eventuelle Cluster verwendet wurde. Insgesamt war für die Tests besonders wichtig ob die minimalen Zeitanforderungen des Anwendungsfalles erfüllt werden und wie zuverlässig das System funktioniert.

durchgeführte Tests Folgende durchzuführende Tests wurden festgelegt:

- Zeitverhalten Nachrichtenauftritt in bekannten Intervallen: Hierbei wurden die Ergebnisse per Zeitdaten festgehalten. Die festgehaltenen Zeitintervalle waren: Verbindungsaufbau, Senden aller Nachrichten, Zeit bis zur Ankunft der ersten/letzten Nachricht. Diese wurden unter den Einflussfaktoren Nachrichtengröße, Heterogenität des Nachrichtenstroms, Anzahl der Verbindungen und Bestätigungsmethode des Consumers getestet. Hier wurden stets die Zeiten des

zweiten Zeitintervalls genommen, um Einfluss durch den direkt davor stattfinden Verbindungsaufbau zu vermeiden und das reine Sendeverhalten beurteilen zu können.

- Bandbreitentest: Es wurde für 10 Minuten ein Nachrichtenstrom über den Broker geschickt. Dieser soll überwältigend für den Empfänger sein um so festzustellen welche Bandbreite der Empfänger maximal bewältigen kann und wie mit den sich ansammelnden Nachrichten in der Queue umgegangen wird. Die Ergebnisse wurden über das Managementtool von RabbitMQ bestimmt und per Screenshot festgehalten.
- Clustertest: hier soll das verhalten der Klienten in einem Cluster von Brokern betrachtet werden und wie sich die Klienten bei Ausfall eines Brokers verhalten. Das Verhalten wurde beobachtet und schriftlich festgehalten.
- TLS/SSL Test: Hierbei wird getestet ob TLS/SSL umsetzbar ist. Ein ausführlicher Test war nicht möglich, da die Implementierung keinen Tests auf dem Realismusgrad der anderen unterstützt hat. In diesem Fall wurde nur festgehalten ob es funktioniert hat oder nicht.

4. Evaluierung

In diesem Kapitel werden zuerst die Ergebnisse der theoretischen Analyse, für die Protokolle betrachtet, gefolgt von den Ergebnissen der praktischen Tests und einer Diskussion und Fehlerbetrachtung aller Ergebnisse.

4.1. Auswertung theoretische Analyse

Da Google Cloud Pub/Sub Service und Amazon Simple Notification Service nicht Teil der Vergleichsanalyse sind, sind sie auch kein Teil der Auswertung. Im Vergleich der anderen Protokolle haben sich allerdings große Unterschiede offenbart. So haben zwar alle Protokolle über 50% der möglichen erreichbaren Punkte geschafft, aber nur 4 Protokolle sind über die 70% Grenze gekommen.
Am besten abgeschnitten hat AMQP mit 74,71%, vor STOMP mit 72,41%, MQTT und Kafka mit 71,26%.
AMQP hat sich besonders dadurch hervorgetan das es viele individuelle Einstellungsmöglichkeiten bietet, die es dem Nutzer ermöglichen es gut an die eigenen Anforderungen anzupassen. Des weiteren gibt es dazu eine lebendige Community, die den Ersteinstieg sowie den Langzeitsupport erleichtert und mit unterstützt[MQ20; Ped19; Goo20a]. Ebenfalls positiv hat sich die Routingfähigkeit mittels hierachischer Topics ausgewirkt[HKM+16]. Abzüge gab es vor allem durch die vergleichsweise hohe Belastung für sowohl Klienten als auch Broker[Moy15] und für die etwas überdurchschnittliche Headergröße[Nai17].
STOMP ist im Vergleich dazu zwar ressourcensparender und ein kleineres, leichteres Protokoll, bietet dafür aber weniger Möglichkeiten. Es ist als Protokoll vor allem soweit oben in der Bewertung gelandet weil STOMP wenig vorschreibt und fast alles vom Benutzer und den Servereinstellungen festgelegt werden kann. Daher wurde der beste Fall bei der Bewertung angenommen. STOMP konnte ebenfalls mit einer lebendigen Community glänzen[Bri09; MLW+18; Goo20f]. Die größten Punktabzüge gab es für wenige bis keine integrierten Einstellungsmöglichkeiten für Sicherheit und QoS, die über TCP hinausgehen[Moy15]. Auch der Mangel an Routingoptionen hat sich negativ niedergeschlagen.
MQTT ist STOMP recht ähnlich, da es ebenfalls ein leichteres Protokoll, als AMQP, ist und dadurch Punktabzug bekommen hat. Dieser entstand, weil diese Leichtigkeit und die geringeren Anforderungen erreicht wurde, durch das weglassen von

Optionen und Einstellungsmöglichkeiten[Moy15]. Zudem ist MQTT das einzige Protokoll das bei Google Trends ein besseres Ergebnis als Kafka erzielt hat[Goo20d]. Trotzdem hat es im Bereich Ersteinstieg und Support, gerade wegen der GitHub-Community[MQT13; Bun12], viele Punkte auf STOMP verloren. und außerdem nur sehr begrenzte Discoverymöglichkeiten unterstützt[Moy15]. Ein weiteres Problem ist, dass MQTT gleichzeitig nicht ganz so frei vom Anwender editierbar ist wie STOMP.
Das letzte Protokoll in den Top 4, Kafka, hat seine großen Vorteile in der Skalierung. Da es das einzige Protokoll ist, dass nativ horizontal skalieren kann[Fou17] und damit große Vorteile für ein großes System bietet. Auch die Geschwindigkeit in der Übertragung und die kleinen Header überzeugen in der Analyse[Fou17]. Allerdings sind die Anforderungen an Client und Broker, ähnlich zu AMQP, recht hoch, die Routing Optionen allerdings im Vergleich begrenzt. Selbst eine gute Bewertung bei Ersteinstieg und Support konnte dies nicht ausgleichen[Fou11]. Letztendlich hat Kafka aber nicht besser abgeschlossen, weil seine Hauptvorteile in den Bereichen Geschwindigkeit und Skalierung bei einer Logistikplattform, mit Anforderungen analog zu dieser Arbeit, nicht Ausschlag gebend sind. Denn diese Stärken sind im Rahmen dieser Arbeit nicht essentiell, da hier nur sehr lose Zeitbeschränkung für die Übertragung und Verarbeitung von Nachrichten erfüllt werden müssen. Wohingegen ein starker Fokus darauf gelegt wird, die Eventnachrichten von verschiedenen Quellen zuverlässig an verschieden Ziele zu übermitteln.
Das nächstbeste Ergebnis hat DDS, mit 66,66% erzielt. Trotz großer Leistungsfähigkeit hat es schlecht abgeschnitten, weil es keine Middleware im klassischen Sinne verwendet und nur sehr begrenzte Möglichkeiten für Ersteinsteiger bietet. Der zweite Punkt kommt besonders zum tragen, da DDS als einziges datenbasiertes System in der Analyse, eine andere Herangehensweise erfordert. Da es kaum Libraries für verschiedene Programmiersprachen gibt und auch die Community nicht so groß und lebendig ist[OC15; Goo20b], wie beispielsweise AQMP oder Kafka, kann daher gerade Neueinsteigern zum Verhängnis werden. Ebenfalls hat hier die Option zur Nutzung von hierarchischen Topics gefehlt. Diese Nachteile konnten auch gute Ergebnisse in den Bereichen der recht geringen Hardwarebelastung und den Einstellungsoptionen für Sicherheit und Qualitiy of Service, nicht ausgleichen. Obwohl DDS gerade bei den QoS Optionen insgesamt am besten abgeschnitten hat[Moy15].
XMPP hat mit 63,21% nur leicht schlechter zu DDS abgeschnitten. Das Problem von XMPP ist, dass es ein Request/Response Protokoll ist das eine Erweiterung für Publish und Subscribe bekommen hat. Was es zu einem vernünftigen Protokoll für unseren Anwendungsfall macht, aber doch Schwächen offenbart, die andere Protokolle, die speziell dafür ausgelegt wurden, nicht haben. So hat es Punkte in der Bewertung verloren weil es in seiner Nachrichten Kodierung weder binär noch String verwendet und somit einen weiteren Kodierungsvorgang benötigt[MSAM20]. Auch sind bei XMPP die Routingoptionen eingeschränkt, sodass es mit AMQP oder MQTT nicht mithalten kann. Auch bietet es keine zusätzlichen Quality of Sevice

Einstellungen[Moy15] und verlässt sich bei der Sicherheit auf das vorhandene Netzwerk und das genutzte Transportprotokoll[IHK18]. XMPP konnte obendrein auch nicht mit einer lebendigen Community überzeugen[Pro13; Pro16; Goo20g]. Die Vorteile bei Hardwarebelastung und Übertragungsgeschwindigkeit konnten die zahlreichen Probleme nicht ausgleichen.
Mit 59,77% fällt SMQ dann noch ein gutes Stück zurück. Was nicht überrascht, da es nicht für ein so großes Softwaresystem wie unsere Hafensoftware ausgelegt ist. SMQ ist das beste Protokoll, unter den Gewählten, zur Kommunikation von und mit IoT-Geräten. Es könnte als Teil einer Plattform hervorragend funktionieren, dann braucht es aber einen Kern rund um ein anderes Protokoll wie AMQP oder Kafka, der per SMQ mit den IoT Endpunkten kommuniziert. Da hier allerdings eine Analyse für ein Gesamtsystem aus nur einem Protokoll vorgenommen wurde ist SMQ dafür ungeeignet. Dazu ist es auch das einzige bewertete proprietäre Protokoll[Moy15]. SMQ bestätigt unsere Vermutung über die Community von proprietären Protokollen aus Unterabschnitt 2.1.1[Log18; Goo20e]. Seine Stärken in Echtzeitanwendungen und seine minimale Hardwarebelastung[Moy15] sind für unseren Anwenderfall nicht Ausschlag gebend, weshalb sie nicht über die Probleme hinweg täuschen können.
Sehr ähnlich zu SMQ ist ZeroMQ, da es auch ein extrem leichtes und schnelles Protokoll ist, das vor allem auf Einfachheit, Geschwindigkeit und Effektivität abzielt. ZeroMQ hat mit 59,77% auch genauso viele Punkte erzielt. Es hat zwar Punkte verloren, weil es keine richtige Middleware verwendet[HKM+16] und dichter an Transportprotokollen ist als an Kommunikationsprotokollen, aber dafür Pluspunkte gesammelt mit seiner Fähigkeit zum horizontalen Skalieren[HKM+16] und seiner lebendigeren Community[pro12; Goo20h].
Den letzten Platz in der Auswertung hat JMS mit 51,72% belegt. Problematisch ist hier, dass es recht groß und hardwarebelastend ist, aber nicht die Einstellungsmöglichkeiten von AMQP bietet und insgesamt in allen Bereichen durchschnittlich ist und keine wirklichen Stärken aufweist, aber deutliche Schwächen[Goo20c].
Insgesamt war es überraschend wie gering die Punkte aller Protokolle ausgefallen sind, da deutlich mehr erwartet wurden. Auch wenn es keine Überraschung ist, dass AMQP, MQTT und Kafka unter den besten wiederzufinden sind, da diese die Bekanntesten und mit die meistbenutzten Protokolle sind. Mit der hohen Bewertung von STOMP wurde nicht gerechnet, da die Vermutung nach der Recherche war, dass AQMP und MQTT die besten sein würden. Außerdem war, im Rahmen des Informatikstudiums, STOMP, noch nicht in relevantem Ausmaß vorgekommen, im Gegenastz zu etwa AMQP oder MQTT. Die Tatsache, dass SMQ und ZeroMQ weit unten gelandet sind hat die Annahmen von vor der Analyse bestätigt, da von vornherein zu erwarten war, dass Sie nicht die Möglichkeiten mitbringen, die für eine Logistikplattform gebraucht werden. Es hat überrascht, dass JMS so schlecht abschneidet, da es vom System her durchaus verwendet wird und auch sinnvoll ist, und ursprünglich vermutet wurde, dass es nur hinter AMQP, MQTT, Kafka zurückfällt. Wahrscheinlich liegt dies daran das JMS eher eine gute Strukturvorlage und API

zur Implementierung einer Middleware ist und kein wirkliches Protokoll. Insgesamt wurden die Vermutungen nach der Recherche bestätigt, da die bekanntesten und populärsten Protokolle die besten Ergebnisse erzielt haben, da STOMP sehr bekannt und viel genutzt ist, auch wenn es der Autor nicht auf dem Schirm hatte.

4.2. Auswertung Tests

Da die Ergebnisse der Tests sehr umfangreich sind und sich sehr ähneln wurde die Entscheidung getroffen hier direkt nur die Ergebnisse der Tests mit heterogenem Nachrichtenstrom zu verwenden, da sie am repräsentativsten für unseren Anwendungsfall sind. Es wurden als Informationen für die ersten Tests die Anzahl an Verbindungen, die Zeit die zum Aufbau aller Verbindungen benötigt wurde, Zeitspanne die das Versenden aller Nachrichten gebraucht hat, die Zeitverzögerung bis zum Erhalt der ersten Nachricht und die Zeitverzögerung bis zum Erhalt der letzten Nachricht aufgenommen. Für die Langzeittests wurde die durchschnittliche Bestätigungsrate des Empfängers betrachtet. Bei Fällen in denen dieser Wert nicht ausgereicht hat um den Verlauf vollständig zu beschreiben, wurden auch noch der Maximalwert und der Minimalwert aufgenommen. Alle Ergebnisse der ersten Tests und der Langzeittests sind bei Interesse aber im Anhang nachzulesen.

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	6,61	0,031253	0,015627	0,37772
250	17	0,147158	0,031249	0,140668
500	36	0,381954	0,015625	0,08285
1000	68	0,735556	0,015624	0,120442
2000	143	1,680029	0,015624	0,665736
4000	297	3,734833	0,040696	1,356109
6000	185	3,989711	0,015628	1,040689
8000	252	2,473246	0,015624	5,174158

Tabelle 4.1.: Zeitdatendes Zeitintervalltest von STOMP, mit heterogenen Nachrichtenstrom und automatischer Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	6	0,0534	0,022149	0,031247
250	14	0,137752	0,022154	0,115916
500	28	0,303919	0,053404	0,265586
1000	64	0,786714	0,015627	0,320196
2000	126	1,852795	0,039521	0.314722
4000	265	3,819601	0,015642	0,795543
6000	189	4,510907	0,05135	5,945539
8000	880	1,328004	0,10901	12,644481

Tabelle 4.2.: Zeitdatendes Zeitintervalltest von STOMP, mit heterogenen Nachrichtenstrom und manueller Nachrichtenbestätigung

STOMP In der Analyse der Zeitdaten aus dem Test, für die Handhabung von festen Nachrichtenmengen zu festen Zeitpunkten, ließ sich keine Abhängigkeit der Leistung von der Größe der Nachrichten oder der Homogenität des Nachrichtenstroms feststellen, da die Zeiten lediglich durch die Anzahl an Verbindungen und zu bewältigen Nachrichten anstieg. Allerdings ließ sich eine starke Abhängigkeit der Leistung vom Arbeitsspeicher des Brokers erkennen, da es immer wieder irreguläre Werte gibt die sich nur durch die Auslastung des Computers, auf dem getestet wurde, erklären lassen (siehe Tabelle 4.1 und Tabelle 4.2).
Allerdings war festzustellen, dass die Zeit zum Aufbau von Verbindungen zwischen Broker und Client exponentiell zugenommen hat, was aber auch mit der wachsenden Auslastung des Brokers und des Computers zusammenhängen dürfte (siehe Tabelle 4.1 und Tabelle 4.2 Spalte 2). Ebenso ließen sich Zeitunterschiede feststellen, wenn das Bestätigen der erhaltenen Nachrichten nicht mehr automatisch passiert, also angenommen wird, dass eine gesendete Nachricht auch angekommen ist, sondern manuell vom Klienten bestätigt werden muss. Wobei es hier aber feststellbar war, dass es mit wachsender Größe der Nachrichten und stärkerer Heterogenität des Nachrichtenstromes zu einer wachsenden Zahl an verlorenen Nachrichten kam. Dies kann auch mit der Implementierung zusammenhängen da die hier gewählte Implementierung die Bestätigung bei Auftreten einer Nachricht vollzogen hat und somit einmal vergessene Nachrichten bis zur Neuverbindung mit dem Broker nicht wieder gefunden hat. Dieses Problem ließe sich aber mit einer anderen überarbeiteten Implementation umgehen und beheben.
Hinsichtlich der Zeitschranken des Anwendungsfalles hat STOMP den ersten Test aber komplett bestanden auch wenn es keine 100 prozentige Zuverlässigkeit garantiert.

Abbildung 4.1.: Ergebnis des Bandbreitentests für STOMP mit heterogenem Nachrichtenstrom und automatischer Bestätigung stellt die Nachrichtenrate des Publishers(gelb) und die Bestätigungsrate des Consumers (grün) dar.

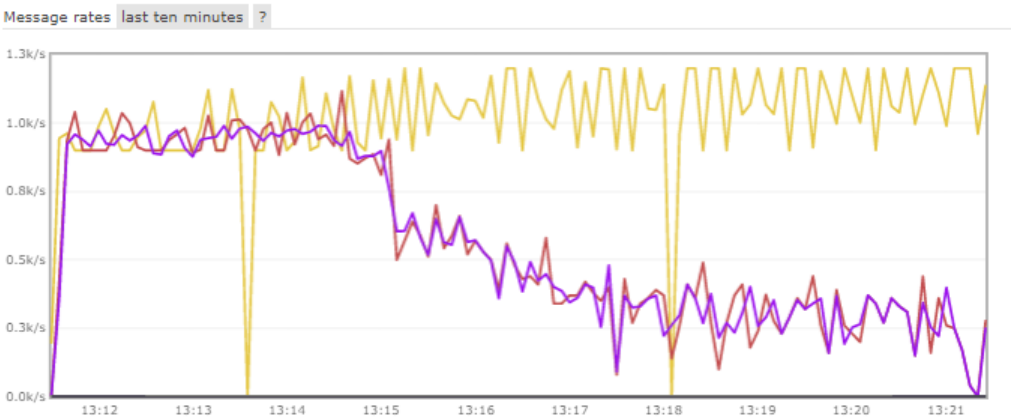

Abbildung 4.2.: Ergebnis des Bandbreitentests für STOMP mit heterogenem Nachrichtenstrom und manueller Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigungsrate des Consumers (lila), Rate des Bestätigungsversand(rot) dar.

Beim zweiten Test zur Untersuchung des Verhaltens bei dauerhaften Nachrichtenströmen haben sich primär 2 Verhaltensweisen für den Nachrichtenverkehr finden lassen. Wenn wir Nachrichten automatisch bestätigen ist ein, von der Anzahl der Verbindungen unabhängiger, Strom von 1200 Nachrichten pro Sekunde erreicht und bestätigt worden (siehe Abbildung 4.1).
Beim manuellem Bestätigen zeigt sich ein anderes Verhalten. Es ist zu Beginn möglich bis zu 900 Nachrichten pro Sekunde zu bestätigen, dann bricht nach im Schnitt

etwa 5 min die Bandbreite ein und pegelt sich bei 200 bis 370 Nachrichten pro Sekunde ein (siehe Abbildung 4.2). Diese Rate wird dann auch konstant gehalten. Zum Zeitpunkt des Einbruches ist der Broker nach eigenen Metriken nicht überlastet, allerdings zeigt der Taskmanager, dass der PC auf dem Broker und Klienten laufen an, dass dieser voll ausgelastet ist, gerade der CPU. Dies zeigt zumindest die Option auf das dieser Einbruch aufgrund begrenzter Hardwareressourcen stattfindet und somit ein Datenstrom von bis zu 900 Nachrichten pro Sekunde möglich ist. Da sich dies aber nicht Testen und belegen ließ, muss vom minimalen Wert ausgegangen werden. Es lässt sich auch kein Zusammenhang zwischen Nachrichtengröße und Bandbreite feststellen da der Test mit den größten Nachrichten die höchste Bandbreite hatte wohingegen der Test mit den im Anwendungsfall primär erwarteten Nachrichten die geringste Bandbreite hatte. Auch die Heterogenität hat sich nicht signifikant ausgezeichnet. Dies legt ebenfalls nahe, dass der maximal mögliche Nachrichtenstrom primär von den verfügbaren Ressourcen abhängt.
Selbst wenn vom schlimmsten Fall ausgegangen wird, gäbe es noch einen Nachrichtenstrom von 200 Nachrichten pro Sekunden. Da jedes Schiff pro Besuch in einem Hafen im Schnitt 3-4 Events auslöst, würde dies dem System pro Knoten das Handhaben von etwa 50 Events pro Sekunde erlauben. Damit ließen sich 72.000 Events pro Tag verarbeiten bzw. 26.280.000 Events im Jahr. Da es im Jahr 2015 aber nur 295.000 Besuche in allen Häfen des Ostseeraumes gab könnte ein Knoten pro Hafenbesuch (Schiff) 89 Events handhaben. Da allerdings nicht von einem Knoten für den ganzen Ostseeraum ausgegangen werden kann, sondern minimal einen Knoten pro Hafen ist sinnvoller das mit Werten für einzelne Häfen zu vergleichen. Die Top 3 Häfen machen 25% des Verkehrs aus, was bedeutet, dass einer dieser Häfen im Schnitt etwa 8,3% des Gesamtverkehrs handhabt. Dies entspricht 24.485 Besuchen im Jahr. Für diese Zahl ermöglicht ein Knoten eine Anzahl von 1073 Events pro Schiff. Dieser Wert lässt sich auch noch durch die Option, mehrere Knoten pro Hafen im Cluster zu verwenden steigern. Somit kann man auch hier davon ausgehen, dass die STOMP Implementierung geeignet wäre, den Anwendungsfall zu bewältigen, solange eine zuverlässigere Implementierung gewählt wird, im Stile eines Pollingservices oder Ähnlichem.
Beim Clustering Test war STOMP nur teilweise erfolgreich. Da das STOMP Plugin von RabbitMQ die Clustering Methode mitbenutzt ist das Einrichten eines Clusters recht einfach. Es wird aber komplizierter und muss manuell selbst implementiert werden, wenn man möchte, dass sich der Klient nach einem Disconnect mit einem anderen Knoten verbindet oder so lange versucht, sich wieder zu verbinden, bis der Knoten wieder verfügbar ist. Leider ließen sich auch keine genauen Daten über das Verhalten eines Clusters bei Netzwerkausfall bestimmen, da das Verhalten bei RabbitMQ über Mehrheitsentscheide geschieht, was in Clustern mit 2 Knoten zu nicht vorhersehbaren und nicht repräsentativen Verhalten führt. Aber Knoten, die in einem Cluster waren, als sie disconnected sind verbinden sich wieder automatisch mit dem Cluster, wenn sie wieder online gehen.

Der 4. Test hat zu verschiedenen Problemen geführt, da sich nicht zuverlässig eine TLS gesicherte Verbindung aufbauen ließ. Auf Basis der RabbitMQ Dokumentation zu TLS[Piv13b] der Broker konfiguriert und mit Hilfe von OpenSSL ein Bündel an Zertifikaten generiert. Woraufhin es zuerst gelang eine erfolgreiche Verbindung aufzubauen und Nachrichten zu versenden. Allerdings ließ sich dieses Ergebnis nach einem Neustart des RabbitMQ Brokers nicht reproduzieren. Da sich die Fehlerursache nicht finden ließ wurde der Test von Beginn an wiederholt mit neu Installiertem Broker und neuen Zertifikaten. Doch auch so ließen sich diese Probleme nicht beheben. Auch nach Ausschalten aller möglich Störfaktoren wie Firewalls, oder Misstrauen gegenüber den Zertifikaten vom Betriebssystem, weigerte sich der Broker eine TLS geschützte Verbindung zu akzeptieren. Die Ergebnisse bei der Recherche zur Fehlerbehebung zeigen allerdings an der Dokumentation keine Fehler vorliegen und legt nahe das die entstanden Fehler an Problemen und Fehlern in der Implementation lagen. Somit ließ sich in diesem Test zwar nicht die TLS Fähigkeit von STOMP belegen, allerdings legen die Erfahrung während des Testes nahe das dies auch mittels RabbitMQ möglich ist.
Das Hauptargument gegen STOMP bleibt aber die Unzuverlässigkeit in der Nachrichtenübertragung. Da es konsequent über alle Testfälle hinweg bei großen Nachrichtenmengen zu Problemen mit nicht erhaltenen und nicht bestätigten Nachrichten kam.

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	1,7	0,029	0,019	0,042
250	3,5	0,039	0,009	0,109
500	6,7	0,03	1,242	1,661
1000	13,6	0,055	3,119	4,198
2000	27	0,183	0,222	10,545
3000	40	0,379	0,508	14,153
4000	51	0,476	1,237	20,46
5000	71	0,72	1,452	29,024

Tabelle 4.3.: Zeitdatendes Zeitintervalltest von AMQP, mit heterogenen Nachrichtenstrom und automatischer Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	2,2	0,024	0,02	0,063
250	3,4	0,038	0,074	0,275
500	7,5	0,039	1,067	1,935
1000	12,1	0,059	0,093	1,977
2000	27	0,212	0,073	9,906
3000	40	0,259	0,806	18,82
4000	51	0,644	3,613	34,833
5000	78	0,805	0,056	43,555

Tabelle 4.4.: Zeitdatendes Zeitintervalltest von AMQP, mit heterogenen Nachrichtenstrom und manueller Nachrichtenbestätigung

AMQP Auch hier ließ sich bei der Analyse der Zeitdaten aus dem 1. Test keine Abhängigkeit der Leistung von der Größe der Nachrichten oder der Homogenität des Nachrichtenstroms feststellen. Auch wenn sich hier der Test nur bis 5000 gleichzeitigen Verbindungen mit je einem Channel durchführen ließ, da bei höheren Werten der CPU des Test PC überlastet wurde und das ganze System einfror. Wie auch schon bei STOMP ließ sich eine starke Abhängigkeit der Leistung vom Arbeitsspeicher des Brokers und der CPU Leistung des PCs erkennen. Da es immer wieder große irreguläre Werte gibt, die sich nur durch die Auslastung des Computers, auf dem getestet wurde, erklären lassen (siehe unter anderem Tabelle 4.3 und Tabelle 4.4).
Ebenfalls war auch hier festzustellen, dass die Zeit zum Aufbau von Verbindungen zwischen Broker und Client exponentiell zugenommen hat, aber deutlich kürzer blieb als bei STOMP. Dies könnte aber auch an der CPU Auslastung des Basissystems liegen, da auch hier aus dem Muster fallende Werte auftreten. Bei AMQP ließen sich allerdings im 1. Test kaum Zeitunterschiede zwischen manueller Bestätigung und automatischer feststellen (siehe Tabelle 4.3 und Tabelle 4.4).

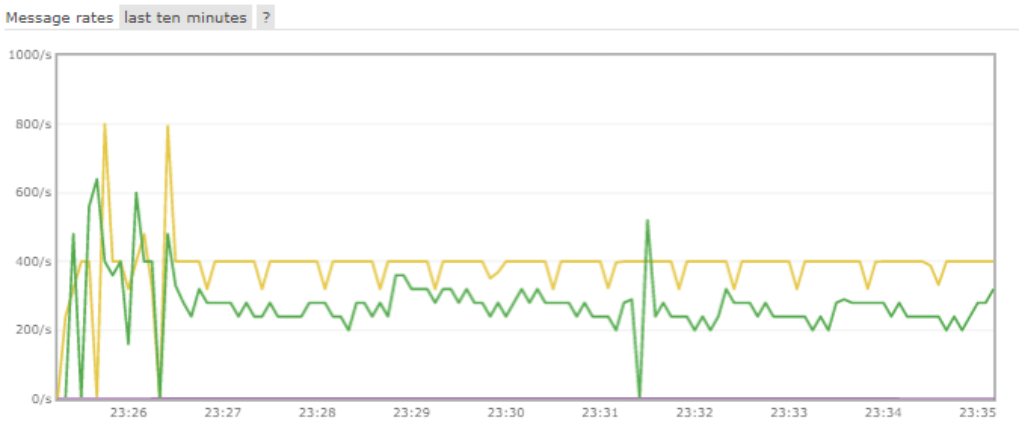

Abbildung 4.3.: Ergebnis des Bandbreitentests für AMQP mit heterogenem Nachrichtenstrom und automatischer Bestätigung stellt den Verlauf der Nachrichtenrate des Publishers (gelb) und die Bestätigungsrate des Consumers (grün) dar.

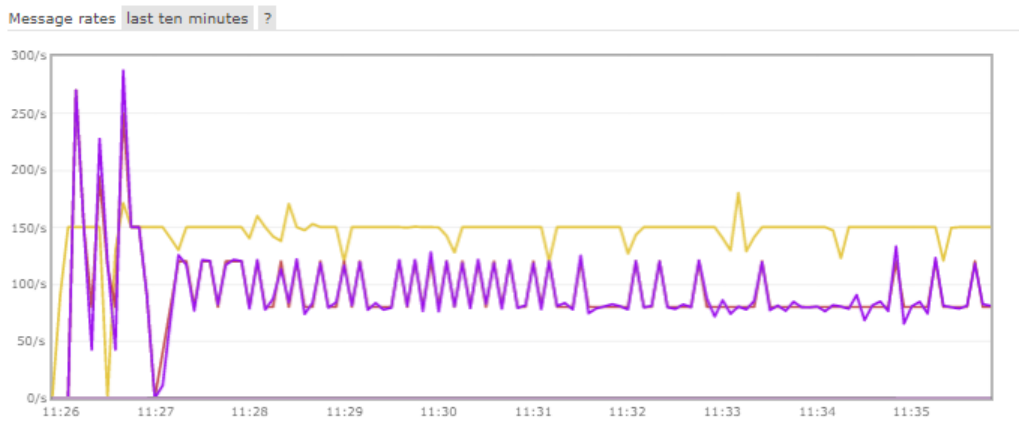

Abbildung 4.4.: Ergebnis des Bandbreitentests für AMQP mit heterogenem Nachrichtenstrom und manueller Bestätigung stellt den Verlauf der Nachrichtenrate des Publishers(gelb) und die Bestätigungsrate des Consumers (lila), Rate des Bestätigungsversand(rot) dar.

Bei der Bandbreitenbestimmung wurden hier allerdings deutlichen Einbußen von im Schnitt etwa 300 Nachrichten pro Sekunde (siehe Abbildung 4.3) auf etwa 90 Nachrichten pro Sekunde (siehe Abbildung 4.4) festgestellt. Bei Betrachtung dieser Bandbreiten ist die geringste dauerhaft gehaltene Bandbreite für AMQP 90 Nachrichten pro Sekunde. Dies ermöglicht 5.400 Nachrichten die Stunde und 129.600 Nachrichten pro Tag. Dies ermöglicht dementsprechend 32.400 Events pro Tag bzw.

11.826.000 Events pro Jahr. Wenn der selben Rechnung wie für STOMP gefolgt wird, ergibt sich für einen Knoten im gesamten Osteeraum ein Wert von 40 Events pro Schiff und runter gerechnet auf einen der Top 3 Häfen 482 Events pro Schiff. Was wenig überraschend einen deutliche Verschlechterung zu STOMP darstellt. Da AMQP allerdings in keinem Test Probleme mit verlorenen und unbestätigten Nachrichten hatte, wäre es hier auch eine Option mit automatischer Bestätigung zu arbeiten. Diese hat eine Bandbreite von schlechtestenfalls 277 Nachrichten pro Sekunde. Damit käme man auf 99.720 Events pro Tag und 36.397.800 Events im Jahr. Somit wäre es möglich mit einem Broker für die Logistikdomäne Ostseeraum immerhin 123 Events pro Hafenbesuch handzuhaben und für einen der Top 3 Häfen heruntergerechnet immerhin schon 1486 Events pro Hafenbesuch. Daher wäre es in diesem Fall für weitere Arbeiten relevant, genau zu evaluieren, welche Klienten unbedingt mit manueller Bestätigung arbeiten müssen und welche nicht. Dies wird von AMQP auch weiter unterstützt da die Filterung und das Routing von Nachrichten an mehrere Queues und Klienten durch die Topics deutlich einfacher gemacht wird. Auch ist es hier deutlich einfacher einen Polling Service zu implementieren.
Der Clustering Test hat bei AMQP bessere Ergebnisse geliefert, da für jeden Verbindungsaufbau eine Liste an Optionen gegeben werden kann, die durchprobiert werden bis eine funktioniert. Auch hat AMQP einen automatischen Reconnectservice immer aktiviert, der im Standardfall schon funktioniert aber noch sehr genau modifiziert werden kann. Insgesamt ist es hier in Verbindung mit RabbitMQ einfacher mit AMQP zu arbeiten, da die gesamte Dokumentation und der Großteil der Communityfragen sich auf AMQP beziehen.
Der TLS/SSL Test hat die selben Ergebnisse und Probleme mit AMQP wie mit STOMP gezeigt. Nur das dies hier noch unerwarteter ist da RabbitMQ ganze Testklienten zur Verfügung stellt um TLS zu entdecken und auszuprobieren. Selbst mit diesen Klienten ließen sich die Probleme nicht beheben. Trotzdem legt auch bei AMQP die Ergebnisse des Testes nahe das TLS mit AMQP und RabbitMQ funktioniert, es aber vermutlich besser wäre Personen mit Erfahrung in diesem Bereich mit diese Implementation zu betrauen.

Insgesamt hat AMQP besser in den Implementationstests abgeschnitten, da es mit besserer Dokumentation der Klientenbibliotheken und dadurch bequemerer Implementation und vor allem deutlich höherer Zuverlässigkeit geglänzt hat. Wohingegen seine deutlichen Nachteile beim Hardwareressourcenverbrauch und Bandbreite für den Anwendungsfall leicht kompensierbar sind. Sei es durch lokale Cluster für einen Hafen, um die Bandbreite zu erhöhen, oder mehr Hardwareressourcen für den Broker. Wie hoch die genauen Hardwareanforderungen für die Klienten sind, lässt sich aus den Tests nicht ermitteln da hier auf einem System der Broker und alle Klienten laufen. Es lässt sich aber vermuten, auf Basis der Geschwindigkeit bei sehr wenigen Verbindungen, dass die Hardwareanforderungen für einen einzelnen Klienten, mit

einer Verbindung, primär den Arbeitsspeicher betreffen, aber noch klein genug sind um auch eine IoT Nutzbarkeit, in der Zukunft, zu ermöglichen. Dies sind aber nur Vermutungen, die in einem weiteren Test geprüft werden müssten und die einzige feste Aussage dazu, die sich aus der Implementationstests ableiten lässt, ist, dass STOMP weniger Hardwareressourcen verbraucht und damit potentiell besser für IoT verwendbar ist. Sowie das STOMP in dieser Implementation länger braucht als AMQP um viele Nachrichten gleichzeitig zu versenden. STOMP benötigt im Schnitt etwa 3,7 Sekunden für 4000 Nachrichten wohingegen AMQP immer unter einer Sekunde braucht. Dafür empfängt STOMP diese 4000 Nachrichten deutlich schneller vom Broker. Bei STOMP betrug der Zeitraum bis zum empfangen der letzten Nachricht zwischen 4,4 Sekunden und 6,4 Sekunden wohingegen es bei AMQP zwischen 14,626 Sekunden und 36,722 Sekunden dauerte. Da diese Verzögerungen allerdings vermutlich auch von der Hardwareleistung abhängen, da sich auch hier Wertschwankungen gezeigt haben, die sich nur so erklären lassen, ist dies kein Indiz um eins der beiden Protokolle zu eliminieren. Da man dadurch nicht schlussfolgern kann, dass der AMQP Client mehr als 120 Sekunden zur Verarbeitung von 8000 Nachrichten braucht. Dies wird auch dadurch bestätigt, dass beim Bandbreitentests die notwendige Bandbreite von 67 Nachrichten pro Sekunde, die für 8000 Nachrichten alle 120 Sekunden notwendig ist, in allen Testfällen erreicht wurde.

4.3. Diskussion/Fehlerbetrachtung

Insgesamt lassen sich an mehreren Stellen Probleme mit den Ergebnissen der theoretischen Auswertung finden. Einerseits gibt es in der Bewertungsmatrix Attributswerte die geschätzt werden primär Hardwareverbrauch/-belastung und Headergröße, da dort keine festen Grenzen festgelegt wurden. Stattdessen wurde eine Klassifizierung in klein, mittel, groß vorgenommen wurde. Dies lässt die Chance für unterbewusstes Beeinflussung der Ergebnisse durch den Auswertenden zu. Es wurde zwar versucht die Kategorisierung komplett neutral vorzunehmen, aber es bleiben dennoch subjektive Einschätzungen die eventuell von Erwartungen beeinflusst sein könnten.
Als zweites Problem fügt sich mit ein, dass es sich an vielen Stellen als schwer bis unmöglich herausgestellt hat genau Zahlenwerte oder konkrete Informationen in der Recherche zu finden. Da viele dieser Werte und Informationen von der Implementierung abhängen, beinhalten die Spezifikationen der Protokolle sehr selten genau Angaben. Dies hat dazu geführt das sehr viele Quellen unterschiedlicher und teilweise mangelhafter Vertrauenswürdigkeit, vor allem Webseiten, in die Recherche eingeflossen sind um eine Einschätzung zu ermöglichen. Dies unterstreicht den Hauptpunkt der bei der Datensammlung beachtet werden muss, nämlich, dass dies eine Informationssammlung für diese Protokolle ist, die gut geeignet ist sich einen Überblick zu verschaffen, die aber keine eigene Recherche zu einem Protokoll, im

Rahmen eines anderen Anwendungsfalles, ersetzen kann.
Ebenso ist gerade die Bewertungsmatrix zwar möglichst allgemein gehalten, sodass sie bei veränderten Gewichtungen noch für andere Anwendungsfälle verwendet werden kann, allerdings war der Anwendungsfall bei Erstellung bereits bekannt und somit kann auch hier unterbewusst eine subjektive Färbung eingeschlichen haben.
Bei der Testimplementierung wurde bereits auf einige der Probleme eingegangen, weshalb sie hier nur kurz genannt werden. Diese Testimplementierung ist nur für jeweils eine Bibliothek des Protokolls, in einer Implementierungsversion vorgenommen und getestet worden. Somit sind die Ergebnisse mehr als Indiz für das Protokoll im Allgemeinen bewertbar. Auch liefen hier alle Klienten und Broker auf dem selben System was zu einer Beeinflussung geführt haben dürfte da sich beide die Ressourcen miteinander teilen mussten. Auch ist dadurch die Frage, wie sehr das Betriebssystem diese Ressourcen Verteilung beeinflusst hat und welche Auswirkung das auf unsere Ergebnisse hatte, entstanden. Sie lässt sich auch nichtmal Ansatzweise beantworten, da eine Untersuchung dieses Faktors den Rahmen dieser Arbeit gesprengt hätte. Dies eröffnet die Möglichkeit zu drastisch anderen Werten bei einer praxisnäheren Testimplementation auf anderen Systemen.
Ebenso ist es möglich, dass besser performende Bibliotheken nicht getestet wurden, da diese für Sprachen verfasst sind, die eine umfangreiche Einarbeitung erfordert hätten. Ein großes Hauptproblem stellt auch da, dass es besser gewesen wäre, alle 4 Protokolle, die über 70% bei der theoretischen Auswertung erreicht haben, in einer Implementation zu testen. Dies war aufgrund von Zeitbeschränkungen der Arbeit nicht möglich. Da in einem Prototypen Kafka bereits implementiert wurde könnten die Erfahrungen daraus als Anhaltspunkt mit verwendet werden, aber für MQTT gibt es keine Informationen. Da Kafka aber von anderen Programmieren implementiert wurde ist es fraglich wie vergleichbar dieser Prototyp mit den Implementationen dieser Arbeit ist. Außerdem gibt es für MQTT nicht mal diese Anhaltspunkte und somit bleibt die Möglichkeit offen, dass die beste Option nicht gefunden wurde.
Auch lässt sich aus den Test nicht bestimmen wie groß der Einfluss der Wahl von RabbitMQ auf die Ergebnisse war. Man kann davon ausgehen, dass selbstgeschriebene Server und Broker schneller und effizienter sind, wie in allen Fällen von spezialisierter Implementierung. Aber selbst wenn nur ein eigener, nicht selbst implementierter Broker für jedes Protokoll verwendet worden wäre, wäre es vermutlich zu drastisch anderen Testergebnissen gekommen. Dies wird weiter dadurch unterstrichen das ActiveMQ von der offiziellen STOMP-Seite als stärkster OpenSource Broker empfohlen wird, nicht RabbitMQ.
Daher gilt auch für den praktischen Testpart was für den theoretischen Part galt: Die Ergebnisse sind nicht sehr allgemein verlässlich sondern als Test zur Bestätigung von Tendenzen zu betrachten. Da zwar versucht wurde alle Faktoren eliminieren, die darin resultieren, dass die Implementation und nicht das Protokoll getestet werden, dies aber nur bedingt gelungen ist. Da AMQP und STOMP zwar über die selbe Middleware getestet wurden und auch die Implementationen im Stil gleich aufge-

baut wurden, aber die Klienten in unterschiedlichen Programmiersprachen verfasst wurden. Was auch einen Einfluss auf die Performance gehabt haben dürfte. Auch war es beim Testen nur bedingt möglich sicherzustellen, dass der zugrundeliegender Windows PC bei allen Tests mit denselben Hintergrundprogramme und Programme, zusätzlich zu RabbitMQ und den Klienten, gleich stark belastet war.
Auch sind diese Implementierungen von TLS Support und Clustering maximal aussagekräftig als Bestätigung der Dokumentation von RabbitMQ, da sie fast genau die Codezeilen aus dieser entsprechen[Piv13b; Piv13a], und somit nicht aussagekräftig für eine realistischere Implementierung sind, da beide Bereiche noch sehr viele Vertiefungen und Einstellungsoptionen bieten die nicht getestet und betrachtet wurden. Speziell TLS kann nicht bewertet werden, da sich in den dortigen Tests keine reproduzierbare Ergebnisse finden ließen. Alle genannten Punkte führen zu dem Schluss, dass die Testergebnisse nicht sehr aussagekräftig sind und maximal zur Überprüfung von Tendenzen aus der theoretischen Analyse genutzt werden sollten, aber auf keinen Fall als endgültige Entscheidungsgrundlage für oder gegen ein Protokoll.

5. Zusammenfassung und Ausblick

Diese Arbeit hat einen Überblick über die bekanntesten Pub/Sub Protokolle des Bereiches geliefert und gibt dem Leser unabhängig vom eigenen Nutzungsbereich einen Anhaltspunkt zur Einschätzung der Protokolle gegeben. Für den speziellen Anwendungsbereich der Logistikdomäne Ostseeraum bietet sie außerdem eine übersichtliche Bewertung dieser Protokolle. Auf Basis des aktuellen Informationsstandes ist die Empfehlung dieser Arbeit für das verwendete Kommunikationsprotokoll AMQP, da es sowohl in der theoretischen Analyse am besten abgeschnitten hat und auch in den Praxistests am besten performed hat. Allerdings bestehenden noch Lücken die gefüllt werden sollten. Daher wäre es sinnvoll auf Basis dieser Arbeit weitere Untersuchungen anzustellen. Im Bereich der Vergleichsanalyse wäre es wichtig ,die Bewertungsmatrix anzupassen und eine Neueinschätzung vorzunehmen, wenn die Eventstrukturen und die Architektur der Logistikplattform besser bekannt sind und sich genauere Aussagen treffen lassen. Im Bereich der praktischen Implementierung wurden hier nur sehr begrenzte Ergebnisse produziert. Besonders wichtig wäre es genauere Hardwareanforderungen zu bestimmen. Damit eine genaue Einschätzung bezüglich der IoT Fähigkeit möglich wird, da dies ein wichtiger Part einer solchen Plattform werden wird, hier aber nur sehr begrenzte Aussagen getroffen werden konnten. Da Cluster voraussichtlich ebenfalls ein essentieller Teil der endgültigen Lösung sein werden, wäre es hier interessant und wichtig, die verschiedenen Clusteringoptionen für AMQP oder andere Protokolle zu vergleichen und welche Leistungssteigerungen ermöglicht werden können, um eine bessere Einschätzung in der Zukunft treffen zu können.

A. Anhang

A.1. Tabellen des Praxistest

A.1.1. STOMP

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	5,892	0,062521	0,015647	0,169307
250	15,1295	0,20018	0,015267	0,269588
500	33	0,736341	0,015622	0,517685
1000	65	0,928432	0,0294614	0,8775315
2000	139	2,440566	0,022139	1,920078
4000	305	3,775959	0,022334	0,633113
6000	182	3,325776	0,031246	1,078527
8000	239	3,734618	0,024996	2,97418

Tabelle A.1.: Zeitdatendes Zeitintervalltests von STOMP, mit minimalen Nachrichtenstrom und automatischer Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	6,506904	0,042719	0,020297	0,144155
250	17	0,115856	0,015581	0,10029
500	35	0,13253	0,02241	0,108522
1000	69	0,815214	0,073171	0,300671
2000	139	1,465117	0,015625	0,090038
4000	303	3,974701	0,015511	0,686117
6000	182	3,290883	0,014995	0,157007
8000	246	2,617816	0,04958	4,830053

Tabelle A.2.: Zeitdatendes Zeitintervalltests von STOMP, mit durchschnittlichem Nachrichtenstrom und automatischer Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	5,398	0,062436	0,01556	0,022143
250	17	0,172696	0,040629	0,087434
500	35	0,351246	0,015445	0,046372
1000	72	0,678414	0,037767	0,200425
2000	141	1,879303	0,022154	1,955775
4000	297	3,78157	0,031248	0,536732
6000	186	3,464817	0,01563	0,250001
8000	252	4,85861	0,046877	2,019372

Tabelle A.3.: Zeitdatendes Zeitintervalltests von STOMP, mit maximalem Nachrichtenstrom und automatischer Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	8	0,069033	0,015627	0,046873
250	16	0,069028	0,053403	0,285205
500	35	0,383475	0,015404	0,221606
1000	68	0,653648	0,015626	0,908469
2000	136	1,660357	0,046879	1,12399
4000	276	3,973333	0,037782	2,467897
6000	187	1,342128	0,015625	8,662393
8000	252	5,814334	0,056001	3,905048

Tabelle A.4.: Zeitdatendes Zeitintervalltests von STOMP, mit minimalem Nachrichtenstrom und manueller Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	7	0,050642	0,022325	0,061557
250	16	0,131521	0,015623	0,147144
500	35	0,337705	0,037779	0,30063
1000	67	0,670334	0,04689	0,53263
2000	129	1,579391	0,037785	1,074365
4000	273	3,7589	0,03125	2,702957
6000	186	3,691985	0,015632	2,05192
8000	252	4,492552	0,031246	9,923535

Tabelle A.5.: Zeitdatendes Zeitintervalltests von STOMP, mit durchschnittlichem Nachrichtenstrom und manueller Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	5	0,015632	0,046884	0,115909
250	16	0,132635	0,053405	0,193086
500	32	0,393953	0,015406	0,260096
1000	66	0,761885	0,015638	0,502023
2000	140	1,769492	0,030422	1,400848
4000	283	3,215153	0,022148	1,603866
6000	188	3,697072	0,031246	2,561915
8000	249	2,372927	0,031248	7,547219

Tabelle A.6.: Zeitdatendes Zeitintervalltests von STOMP, mit maximalem Nachrichtenstrom und manueller Nachrichtenbestätigung

A.1.2. AMQP

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	8	0,022	0,009	0,145
250	14	0,06	0,899	1,315
500	31	0,138	0,027	0,756
1000	59	0,161	0,217	2,486
2000	104	0,187	12,73	19,986
3000	132	0,377	0,012	21,253
4000	189	0,534	0,86	30,391
5000	176	0,703	0,245	43,205

Tabelle A.7.: Zeitdatendes Zeitintervalltest von AMQP, mit minimalen Nachrichtenstrom und automatischer Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	1,8	0,031	0,013	0,085
250	3,6	0,026	2,569	2,712
500	7,4	0,096	0,024	0,223
1000	13,6	0,137	16,325	17,603
2000	25,2	0,147	0,568	6,174
3000	42	0,269	0,007	12,428
4000	53	0,514	1,273	23,555
5000	74	0,756	1,492	28,087

Tabelle A.8.: Zeitdatendes Zeitintervalltest von AMQP, mit durchschnittlichem Nachrichtenstrom und automatischer Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	1,9	0,01	0,104	0,155
250	3,6	0,018	1,183	1,289
500	6,8	0,052	1,417	1,827
1000	14	0,146	0,007	1,3
2000	26	0,251	1,315	6,782
3000	42	0,375	2,617	16,405
4000	54	0,868	0,009	22,181
5000	72	0,565	0,044	28,738

Tabelle A.9.: Zeitdatendes Zeitintervalltest von AMQP, mit maximalem Nachrichtenstrom und automatischer Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	11	0,022	0,081	0,335
250	30	0,081	0,083	0,661
500	25	0,153	0,012	0,89
1000	70	0,273	0,006	4,378
2000	143	0,456	1,704	22,259
3000	52	0,443	0,783	8,474
4000	53	0,969	0,009	13,656
5000	76	0,883	0,085	19,968

Tabelle A.10.: Zeitdatendes Zeitintervalltest von AMQP, mit minimalem Nachrichtenstrom und manueller Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	8,8	0,032	0,072	0,282
250	19	0,073	0,313	1,073
500	38	0,091	0,154	2,564
1000	107	0,293	0,009	2,796
2000	26	0,183	0,1	9,502
3000	41	0,431	1,016	18,515
4000	52	0,497	16,702	36,225
5000	72	0,458	5,63	37,587

Tabelle A.11.: Zeitdatendes Zeitintervalltest von AMQP, mit durchschnittlichem Nachrichtenstrom und manueller Nachrichtenbestätigung

Anz.Verb.	Verb.aufb./s	Versandzeit/s	Verz.1.Nachr./s	Gesamtverz./s
100	1,8	0,028	0,007	0,075
250	3,7	0,013	11,386	11,616
500	7,1	0,047	3,418	3,846
1000	13	0,08	0,009	2,525
2000	26	0,259	0,002	6,893
3000	38	0,272	3,604	12,124
4000	53	0,583	4,11	26,576
5000	71	0,525	1,04	27,347

Tabelle A.12.: Zeitdatendes Zeitintervalltest von AMQP, mit maximalem Nachrichtenstrom und manueller Nachrichtenbestätigung

A.2. Bilder der Langzeittests

A.2.1. AMQP

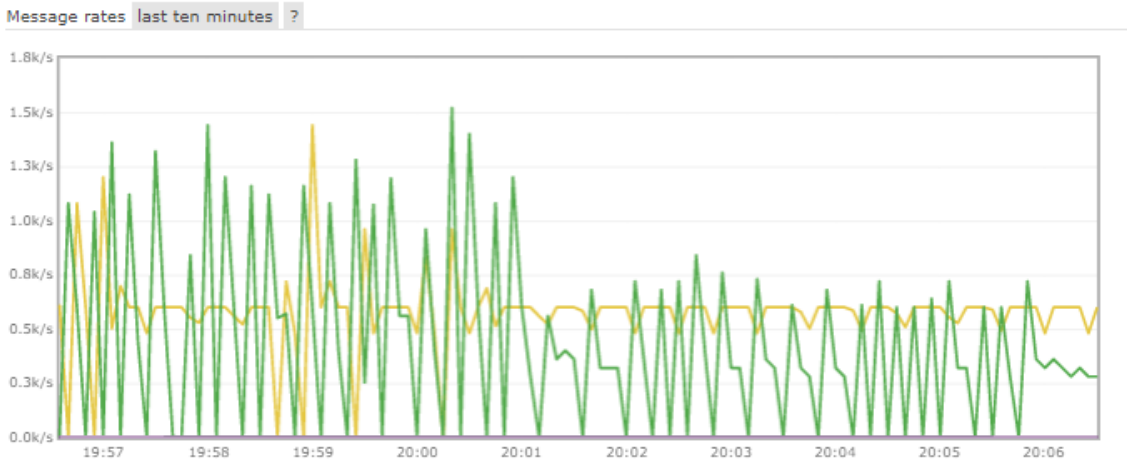

Abbildung A.1.: Ergebnis des Bandbreitentests für AMQP mit minimalem Nachrichtenstrom und automatischer Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (grün) dar. Ergebnisse über die 10 Minuten: durchschnittliche Rate von 368 Nachrichten/s.

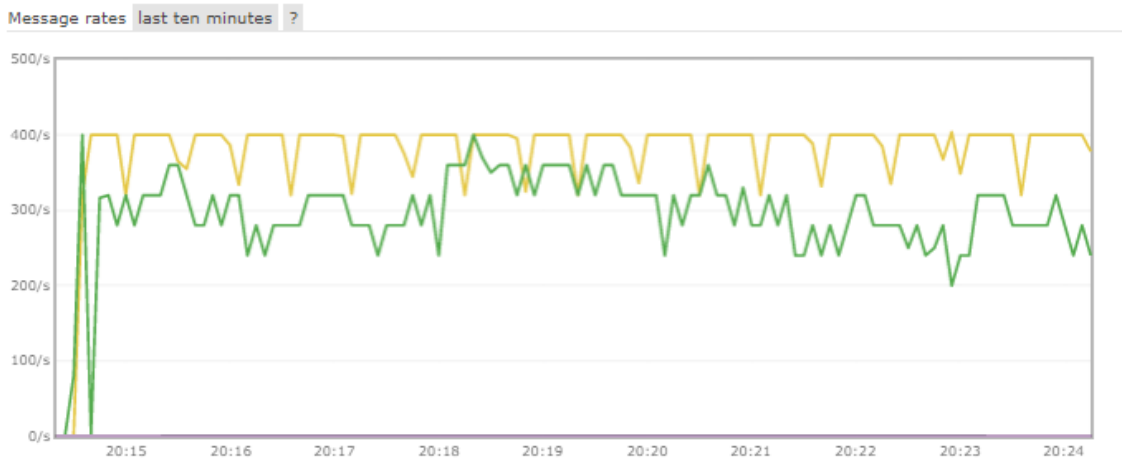

Abbildung A.2.: Ergebnis des Bandbreitentests für AMQP mit maximalen Nachrichtenstrom und automatischer Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (grün) dar. Ergebnisse über die 10 Minuten: durchschnittliche Rate von 290 Nachrichten/s.

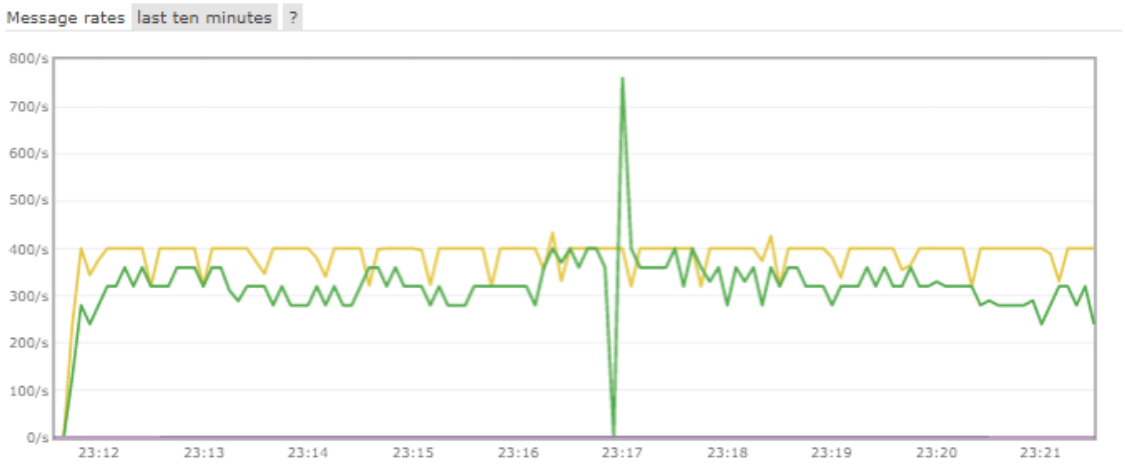

Abbildung A.3.: Ergebnis des Bandbreitentests für AMQP mit maximalen Nachrichtenstrom und automatischer Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (grün) dar. Ergebnisse über die 10 Minuten: durchschnittliche Rate von 310 Nachrichten/s.

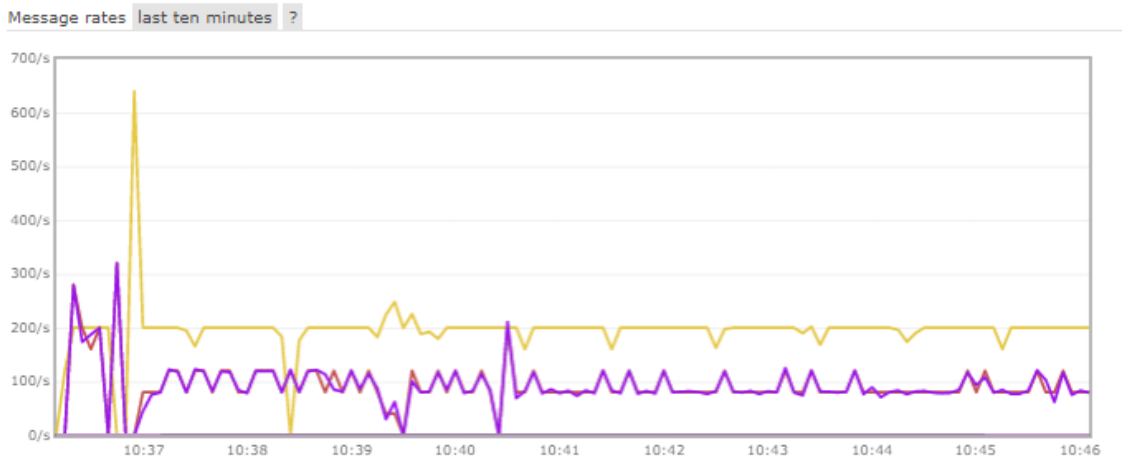

Abbildung A.4.: Ergebnis des Bandbreitentests für AMQP mit minimalem Nachrichtenstrom und manueller Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (lila), Rate des Bestätigungsversand(rot) dar. Ergebnisse über die 10 Minuten: durchschnittliche Rate von 94 Nachrichten/s.

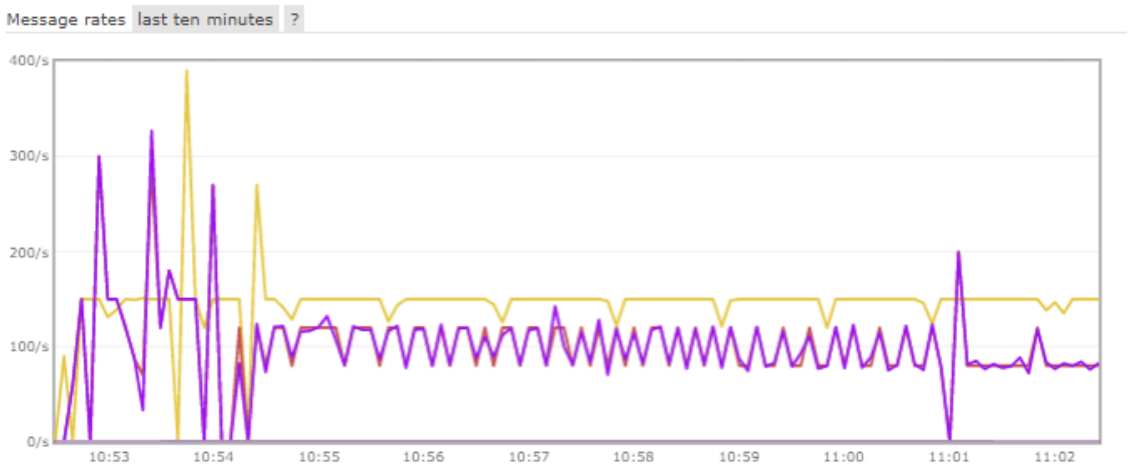

Abbildung A.5.: Ergebnis des Bandbreitentests für AMQP mit durchschnittlichen Nachrichtenstrom und manueller Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (lila), Rate des Bestätigungsversand(rot) dar. Ergebnisse über die 10 Minuten: durchschnittliche Rate von 97 Nachrichten/s.

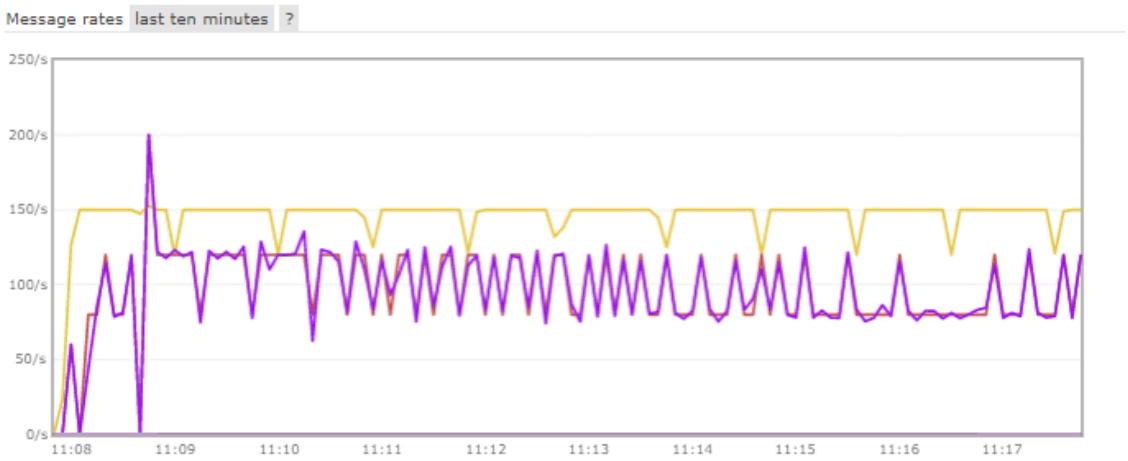

Abbildung A.6.: Ergebnis des Bandbreitentests für AMQP mit maximalen Nachrichtenstrom und manueller Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (lila), Rate des Bestätigungsversand(rot) dar. Ergebnisse über die 10 Minuten: durchschnittliche Rate von 97 Nachrichten/s.

A.2.2. STOMP

Abbildung A.7.: Ergebnis des Bandbreitentests für STOMP mit minimalem Nachrichtenstrom und automatischer Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (grün) dar. Ergebnisse über die 10 Minuten: durchschnittliche Rate von 1193 Nachrichten/s.

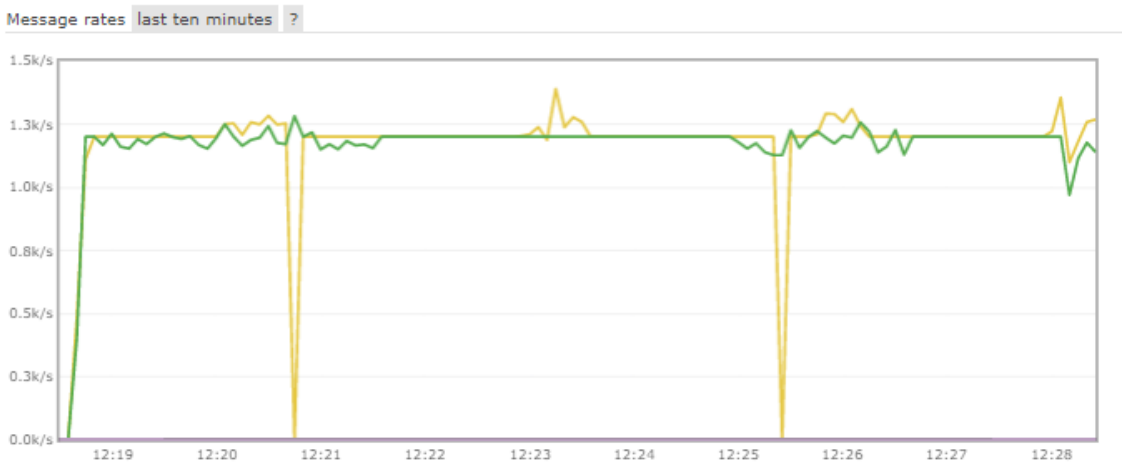

Abbildung A.8.: Ergebnis des Bandbreitentests für STOMP mit durchschnittlichem Nachrichtenstrom und automatischer Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (grün) dar. Ergebnisse über die 10 Minuten: durchschnittliche Rate von 1200 Nachrichten/s.

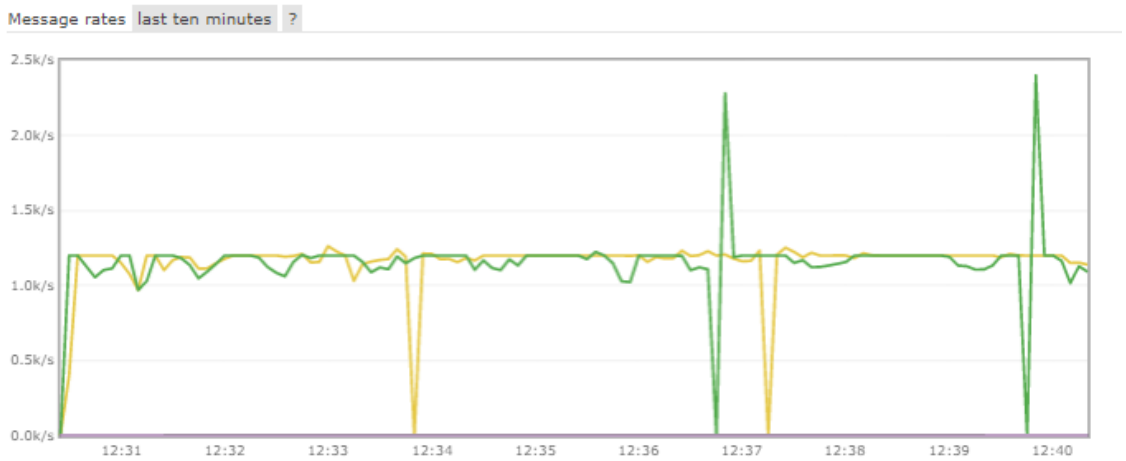

Abbildung A.9.: Ergebnis des Bandbreitentests für STOMP mit maximalen Nachrichtenstrom und automatischer Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (grün) dar. Ergebnisse über die 10 Minuten: durchschnittliche Rate von 1164 Nachrichten/s.

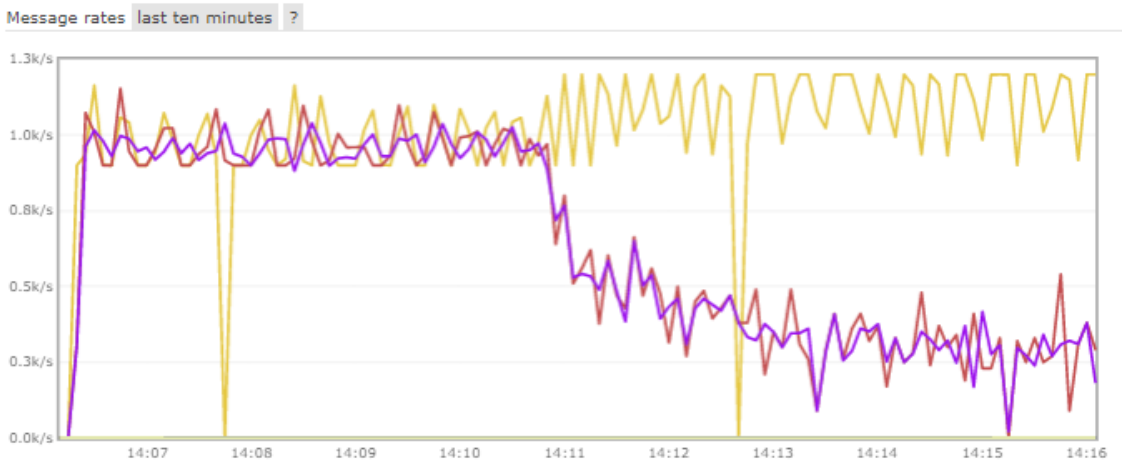

Abbildung A.10.: Ergebnis des Bandbreitentests für STOMP mit minimalem Nachrichtenstrom und manueller Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (lila), Rate des Bestätigungsversand(rot) dar. Ergebnisse über die 10 Minuten: maximale Rate von 1000 Nachrichten/s , minimale Rate von 250 Nachrichten/s, durchschnittliche Rate von 600 Nachrichten/s.

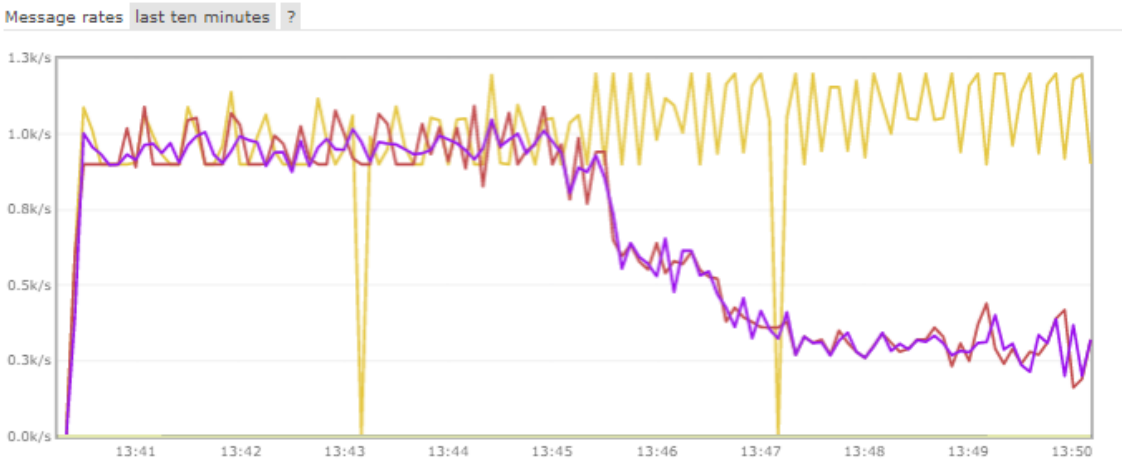

Abbildung A.11.: Ergebnis des Bandbreitentests für STOMP mit durchschnittlichem Nachrichtenstrom und manueller Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (lila), Rate des Bestätigungsversand(rot) dar. Ergebnisse über die 10 Minuten: maximale Rate von 900 Nachrichten/s, minimale Rate von 200 Nachrichten/s, durchschnittliche Rate von 658 Nachrichten/s.

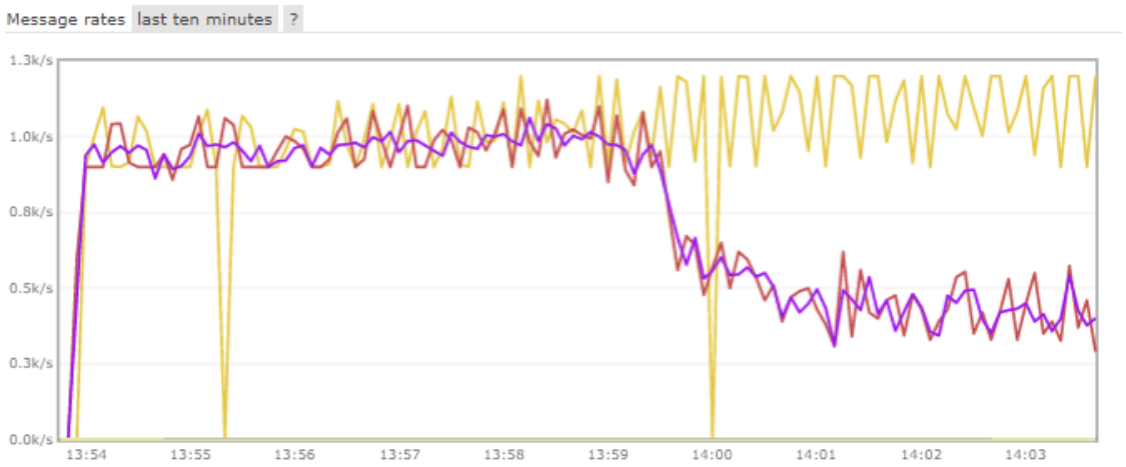

Abbildung A.12.: Ergebnis des Bandbreitentests für STOMP mit maximalen Nachrichtenstrom und manueller Bestätigung stellt die Nachrichtenrate des Publishers(gelb), die Bestätigunsrate des Consumers (lila), Rate des Bestätigungsversand(rot) dar. Ergebnisse über die 10 Minuten: maximale Rate von 971 Nachrichten/s, minimale Rate von 370 Nachrichten/s, durchschnittliche Rate von 700 Nachrichten/s.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

IoT	Internet of Things
MOM	Message Oriented Middleware
AMQP	Advanced Message Queing Protokoll
MQTT	Message Queing Telemetry Transport Protokoll
STOMP	Streaming Text-Oriented Message Protokoll
XMPP	Extensible Messaging Presence Protocol
DDS	Data Distribution Service
SMQ	Simple Message Queue
JMS	Java Messaging Service
QoS	Quality of Service

Literaturverzeichnis

BPM16 Babovic, Zoran B. ; Protic, Jelica ; Milutinovic, Veljko: Web Performance Evaluation for Internet of Things Applications. In: *IEEE Access* 4 (2016), S. 6974–6992

Bra Braden, Robert T.: *Requirements for Internet Hosts - Application and Support*. RFC 1123, Oktober

Bra89 Braden, Robert T.: *Requirements for Internet Hosts - Communication Layers*. RFC 1122. `https://rfc-editor.org/rfc/rfc1122.txt`. Version: Oktober 1989 (Request for Comments)

Bri09 Briggs, Jason R.: *stomp.py*. `https://github.com/jasonrbriggs/stomp.py/tree/dev`, 2009. – [Online, Zugriff am 09.07.2020]

Bun12 Buntsevich, Adam Rudd; Matteo Collina; Maxime Agor; S.: *MQTT.js*. `https://github.com/mqttjs/MQTT.js/graphs/commit-activity`, 2012. – [Online, Zugriff am 09.07.2020]

CI12 Chirino, Hiram ; Inc., Red H.: *STOMP Protocol Specification, Version 1.2*. `https://stomp.github.io/stomp-specification-1.2.html#Protocol_Overview`, 2012. – [Online, Zugriff am 09.07.2020]

CK16 Chen, Yuang ; Kunz, Thomas: Performance evaluation of IoT protocols under a constrained wireless access network. In: *2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT)*, IEEE, apr 2016

Fou11 Foundation, Apache S.: *Apache Kafka*. `https://github.com/apache/kafka/graphs/commit-activity`, 2011. – [Online, Zugriff am 09.07.2020]

Fou17 Foundation, Apache S.: *Introduction*. `https://kafka.apache.org/documentation.html`, 2017. – [Online, Zugriff am 09.07.2020]

Goo20a Google: *Google Trends AMQP, Kafka*. `https://trends.google.de/trends/explore?geo=DE&q=/m/0dyn96,/m/0zmynvd`, 2020. – [Online, Zugriff am 01.04.2020]

Goo20b Google: *Google Trends DDS, Kafka*. https://trends.google.de/trends/explore?geo=DE&q=/m/06b00w,/m/0zmynvd, 2020. – [Online, Zugriff am 01.04.2020]

Goo20c Google: *Google Trends JMS, Kafka*. https://trends.google.de/trends/explore?geo=DE&q=/m/0bs6g,/m/0zmynvd, 2020. – [Online, Zugriff am 01.04.2020]

Goo20d Google: *Google Trends MQTT, Kafka*. https://trends.google.de/trends/explore?geo=DE&q=/m/0h5619c,/m/0zmynvd, 2020. – [Online, Zugriff am 01.04.2020]

Goo20e Google: *Google Trends SMQ, Kafka*. https://trends.google.de/trends/explore?geo=DE&q=SMQ,/m/0zmynvd, 2020. – [Online, Zugriff am 01.04.2020]

Goo20f Google: *Google Trends STOMP, Kafka*. https://trends.google.de/trends/explore?geo=DE&q=/m/0281q22,/m/0zmynvd, 2020. – [Online, Zugriff am 01.04.2020]

Goo20g Google: *Google Trends XMPP, Kafka*. https://trends.google.de/trends/explore?geo=DE&q=/m/01bb8j,/m/0zmynvd, 2020. – [Online, Zugriff am 01.04.2020]

Goo20h Google: *Google Trends ZeroMQ, Kafka*. https://trends.google.de/trends/explore?geo=DE&q=/m/0cm944w,/m/0zmynvd, 2020. – [Online, Zugriff am 01.04.2020]

Goo20i Google: *Pub/Sub: Skalierbarer Messaging-Dienst für Google*. https://cloud.google.com/pubsub/architecture?hl=de, 2020. – [Online, Zugriff am 09.07.2020]

Gro15 Group, Object M.: *Data Distribution Service (DDS) Version 1.4*. https://www.omg.org/spec/DDS/1.4/PDF, 2015. – [Online, Zugriff am 09.07.2020]

Hin13 Hintjens, Pieter: *ZeroMQ*. http://zguide.zeromq.org/page:all, 2013. – [Online, Zugriff am 09.07.2020]

HKM+16 Happ, Daniel ; Karowski, Niels ; Menzel, Thomas ; Handziski, Vlado ; Wolisz, Adam: Meeting IoT platform requirements with open pub/sub solutions. In: *Annals of Telecommunications* 72 (2016), jul, Nr. 1-2, S. 41–52

HW16 Happ, Daniel ; Wolisz, Adam: Limitations of the Pub/Sub pattern for cloud based IoT and their implications. In: *2016 Cloudification of the Internet of Things (CIoT)*, IEEE, nov 2016

IHK18 Ismail, Ahmed A. ; Hamza, Haitham S. ; Kotb, Amira M.: Performance Evaluation of Open Source IoT Platforms. In: *2018 IEEE Global Conference on Internet of Things (GCIoT)*, IEEE, dec 2018

KRD11 Kumar ; Rakesh ; Dave, Mayank: A Comparative Study of Various Routing Protocols in VANET. In: *International Journal of Computer Science Issues* 8 (2011)

Kre17 Kreps, Jay: *A Guide To The Kafka Protocol*. https://cwiki.apache.org/confluence/display/KAFKA/AGuideToTheKafkaProtocol, 2017. – [Online, Zugriff am 09.07.2020]

Log18 Logic, Real T.: *SMQ C Client Library*. https://github.com/RealTimeLogic/SMQ/graphs/commit-activity, 2018. – [Online, Zugriff am 09.07.2020]

Mah04 Mahmoud, Qusay H.: *Getting Started with Java Message Service (JMS)*. https://www.oracle.com/technical-resources/articles/java/intro-java-message-service.html, 2004. – [Online, Zugriff am 09.07.2020]

MLW+18 Mesnil, Jeff ; Lindsay, Jeff ; Williams, Vanessa ; Kumar, Deepak ; Deep, Astha ; Sellars, Dillon ; Charalampidis, Jimi ; Georgiev, Dimitar ; Raul: *STOMP.js*. https://github.com/stomp-js/stompjs/graphs/commit-activity, 2018. – [Online, Zugriff am 09.07.2020]

Moy15 Moyer, Bryon: *All About Messaging Protocols - What Are the Differences?* https://www.eejournal.com/article/20150420-protocols/, 2015. – [Online, Zugriff am 09.07.2020]

MQ20 MQ, Rabbit: *RabbitMQ*. https://github.com/rabbitmq, 2020. – [Online, Zugriff am 09.07.2020]

MQT13 MQTT.org: *mqtt.github.io*. https://github.com/mqtt/mqtt.github.io/graphs/commit-activity, 2013. – [Online, Zugriff am 09.07.2020]

MSAM20 Millard, Peter ; Saint-Andre, Peter ; Meijer, Ralph: *XEP-0060: Publish-Subscribe*. https://xmpp.org/extensions/xep-0060.html, 2020. – [Online, Zugriff am 09.07.2020]

Nai17 Naik, Nitin: Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: *2017 IEEE International Systems Engineering Symposium (ISSE)*, IEEE, oct 2017

OC15 Object Computing, Incorporated: *OpenDDS*. https://github.com/objectcomputing/OpenDDS/graphs/commit-activity, 2015. – [Online, Zugriff am 09.07.2020]

Org88 Organisation, International M.: *15th SESSION 1987 (Resolutions 596-635)*. https://books.google.de/books?id=w74-3TbkCAkC&pg=PA17&lpg=PA17&redir_esc=y#v=onepage&q&f=false, 1988. – [Online, Zugriff am 09.07.2020]

Org20 Organisation, International M.: *IMO identification number schemes*. http://www.imo.org/en/OurWork/MSAS/Pages/IMO-identification-number-scheme.aspx, 2020. – [Online, Zugriff am 09.07.2020]

Ped19 Pederson, Barry: *Python AMQP 0.9.1 client library*. https://github.com/celery/py-amqp/graphs/commit-activity, 2019. – [Online, Zugriff am 09.07.2020]

Piv13a Pivotal: *Clustering Guide*. https://www.rabbitmq.com/clustering.html, 2013. – [Online, Zugriff am 09.07.2020]

Piv13b Pivotal: *TLS Support*. https://www.rabbitmq.com/ssl.html, 2013. – [Online, Zugriff am 09.07.2020]

pro12 project, The Z.: *Official Github*. https://github.com/zeromq, 2012. – [Online, Zugriff am 01.02.2020]

Pro13 ProcessOne: *ejabberd Community Edition*. https://github.com/processone/ejabberd/graphs/commit-activity, 2013. – [Online, Zugriff am 09.07.2020]

Pro16 ProcessOne: *Erlang/Elixir XMPP library*. https://github.com/processone/xmpp/graphs/commit-activity, 2016. – [Online, Zugriff am 09.07.2020]

SAKB10 Sachs, Kai ; Appel, Stefan ; Kounev, Samuel ; Buchmann, Alejandro: Benchmarking Publish/Subscribe-Based Messaging Systems. In: *Database Systems for Advanced Applications*. Springer Berlin Heidelberg, 2010, S. 203–214

Ser20 Services, Amazon W.: *Amazon Simple Notification Service: Developer Guide*. https://docs.aws.amazon.com/sns/latest/dg/sns-dg.pdf$#$welcome, 2020. – [Online, Zugriff am 09.07.2020]

Sha08 Sharp, Robin: *Principles of Protocol Design*. Springer Berlin Heidelberg, 2008

Sys16 Systems, SDC: *SMQ Broker*. https://www.sdcsystems.com/embedded/real-time-logic/barracuda-application-server/smq-broker/, 2016. – [Online, Zugriff am 09.07.2020]