TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät für Informatik Institut für Systemarchitektur, Professur für Rechnernetze

Diplomarbeit

PROTOKOLLE ZUR BEREITSTELLUNG VON PERIODISCH ERFASSTEN DATEN FÜR MOBILE ENDGERÄTE MIT VARIABLEN QUALITÄTSMERKMALEN

Tristan Heinig

Matrikelnummer: 3404474

Betreut durch:
Dr.-Ing. Thomas Springer

Verantwortlicher Hochschullehrer:
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Eingereicht am 30. Juni 2014

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen wörtlich oder sinngemäß übernommenen Gedanken sind als solche kenntlich gemacht. Ich erkläre ferner, dass ich die vorliegende Arbeit an keiner anderen Stelle als Prüfungsarbeit eingereicht habe oder einreichen werde.

Dresden, 30. Juni 2014

INHALTSVERZEICHNIS

1 EINLEITUNG

1.1 EINFÜHRUNG IN DAS THEMENGEBIET

1.2 MOTIVATION

Fahrzeuge sind bereits heute mit einer Vielzahl von elektronischen Steuereinheiten ausgerüstet. Vorrangig dienen diese der Regelung und Analyse komplexer Vorgänge innerhalb der Fahrzeugmechanik und –elektronik. In Zukunft bleiben diese Daten nicht wie bisher im Fahrzeug selbst gekapselt, sondern nehmen ihren Weg über Kommunikationskanäle zu entfernten Dienstschnittstellen. Voraussetzung dafür sind Verbindungen in übergeordnete Netze, nicht zuletzt ins Internet. Es ist offensichtlich, dass hierbei drahtlose Übertragungsmechanismen zum Einsatz kommen. Mit der Anbindung ans Netz ergeben sich schlagartig neue Möglichkeiten, die insbesondere Interaktionen mit dem Fahrer erlauben.

Zusammenhänge sollen durch eine Auswahl bereits heute realistischer Szenarien verdeutlicht werden.

Sensorgestütze Parkplatzsuche

Die Problematik in Großstädten ist bekannt. In dicht besiedelten Gebieten sind Parkplätze knapp. Verschärft wird dieses Problem, wenn es sich darüber hinaus um attraktive Gebiete für Auswärtige handelt, wie Einkaufs- und Gewerbezentren. Ferner sind städtebauliche Konzepte aus früheren Zeiten nicht für die heutige Zahl von Fahrzeugen geeignet. Nicht nur aus Kostengründen ist die effektive Nutzung zur Verführung stehender Stellplätze relevant. Eine Lösung mit Hilfe von Sensoren soll zeigen, wie Informationen über freie Plätze sinnvoll verteilt werden können.

Die Information darüber, ob ein Parkplatz frei oder besetzt ist, wird über Sensoren im Boden festgestellt, welche in periodischen Abständen Zustandsinformationen an eine zentrale Registrierungsstelle schicken. Für Endkunden liefert eine Dienstschnittstelle Echtzeitinformationen über nahe gelegene Parkmöglichkeiten. Ein intelligenter Algorithmus regelt die Verteilung der Fahrzeuge im Stadtgebiet. Neben den Sensordaten übermitteln beteiligte Fahrzeuge ihre Positionsdaten sowie Fahrzeugeigenschaften, zum Beispiel deren Länge. Als Ergebnis erhalten Anfragende eine Liste mit zur Verfügung stehenden Stellplätzen. Wird ein Platz belegt oder frei, meldet der dort installierte Sensor die Änderung und interessierte Instanzen werden benachrichtigt.

Notfalladministration

Konzepte vernetzter Strukturen in Notfallsituationen sind heute schon Bestandteil vieler Szenarien [JiAn10]. Das Potenzial reicht von der Aufzeichnung seismischer Aktivitäten zur Erdbebenvorhersage [AlSh09] bis hin zur Patientenüberwachung von vitalen Funktionen [00a]. Das folgende Beispiel fügt sich gleichermaßen in den Gegenstand der Arbeit.

Bei einem Notfall und den häufig damit verbundenen Krankentransport überwacht ein ausgebildeter Arzt oder Sanitäter den zu versorgenden Patienten. Es kommen verschiedenste Messinstrumente zum Einsatz. Angenommen das Fahrzeug verfügt über eine Datenverbindung zur Notfallleitstelle, respektive zum Krankenhaus, dann können die aufgezeichneten Messdaten direkt dorthin

gesendet werden. Die zusätzlichen Informationen erlauben frühzeitige Vorbereitungen. Die verantwortlichen Ärzte sind bereits während der Fahrt über den Zustand des Patienten informiert. Ferner kann die Leitstelle die Fahrt des Krankenwagens nachvollziehen und dessen Eintreffen bestimmen, gegebenenfalls sogar eine optimale Fahrtroute hinsichtlich der Verkehrslage vorschlagen.

Elektromobilitätsmanagement

Das letzte Beispiel stammt aus dem Bereich Elektromobilität. Ein zentrales Aufgabenfeld betrachtet Varianten von Versorgungsinfrastrukturen mit Ladestellen. Im Vergleich zu konventionellen Kraftstoffantrieben gibt es bei Elektrofahrzeugen zurzeit noch deutliche Beschränkungen in der Reichweite. Hinzu kommen längere Ladezeiten sowie kein flächendeckendes Angebot von öffentlichen Ladestellen. Umso wichtiger ist es vorhandene Strukturen effizient und kundengerecht zu etablieren.

Eine Vernetzung der einzelnen kooperierenden Entitäten bildet die Basis für eine intelligente Nutzung und zielführende Steuerung. Dazu gehören alle Arten öffentlich zugänglicher Ladestationen, deren Betreiber sowie Elektrofahrzeuge und deren Halter. Sensoren erfassen die Kapazität der Fahrzeugbatterien und errechnen die verbleibende Reichweite. Auf Stationsseite werden Belegung, Ladegeschwindigkeit, Preis und weitere notwendige Informationen erfasst. Eine Schnittstelle im Netz aggregiert die periodisch empfangenen Daten und bereitet sie sinnvoll für die Nutzer auf. Als Antwort erhalten Fahrer eine Übersicht von verfügbaren Ladestationen in ihrer Nähe, mitsamt den Einzelheiten für einen Ladevorgang.

Die vorgestellten Szenarien sollen als Leitfaden für die folgende Arbeit dienen. Dabei werden zentrale Fragen abgeleitet, welche die Grundlage der Betrachtung bilden. Natürlich können nicht alle Fragen in vollem Umfang bearbeitet werden. Einen groben Rahmen bildet das eingangs vorgestellte Thema. Am Anfang erfolgen eine flächige Analyse der relevanten Teilbereiche, sowie eine Bewertung, welches Thema sich für eine tiefere Betrachtung eignet, mitsamt der Abgrenzung von ungeeigneten Feldern. Zu den vorgestellten Beispielen lassen sich folgende Fragen formulieren.

- Wer überträgt?
- Wo/Wohin wird übertragen?
- Wie wird übertragen?
- Wann wird übertragen?
- Was wird übertragen?

Die Analyse beschäftigt sich mit Sendern und Empfängern von Daten. Es zeigt sich, dass diese Rollenverteilung nicht statisch ist und durch eine *Übermittlerrolle* erweitert werden muss. Nicht in allen Fällen überträgt der Sender direkt an den Empfänger und eine Kopplung von Sender und Empfänger ist nicht immer gegeben. Die Klärung von *Wer* und *Wohin* ist oftmals direkt abhängig von der Kommunikationstechnik und damit *Wie* übertragen wird. Die Grundlage bilden dabei Protokolle unterschiedlichster Ausprägung. Die Kommunikation zwischen Geräten erfordert die beiderseitige Einigung auf ein bestimmtes Format. Die Menge von Regeln, die ein solches Format beschreibt, nennt man Protokoll [14a]. Dies ist eine allgemeine Definition und es gibt unterschiedliche Protokolle für unterschiedliche Aufgaben innerhalb eines Kommunikationskanals, die sich gegenseitig beeinflussen. Man arbeitet daher mit einem Protokollstapel. Ferner ist es von großer Bedeutung, ob es sich um einer drahtgebundene oder drahtlose Übertragung handelt. Beide Varianten wirken sich ebenso auf die Frage aus, *Wann* gesendet wird. Eine drahtlose Verbindung kann bisweilen unterbrochen

sein. Auch kann die Anwendung eine Kommunikation in Echtzeit verlangen. Diese Arbeit fokussiert insbesondere periodische Daten, die einige Besonderheiten aufweisen. Zusammen mit einer Analyse der zu übertragenden Daten können Optimierungen auf der Anwendungs-/Datenebene getroffen werden.

1.3 Gliederung der Arbeit

Anlehnend an den formulierten Fragen soll nun der Umfang der Arbeit näher skizziert werden. Das folgende Grundlagenkapitel enthält eine Auswahl verwandter Arbeiten, die sich im Untersuchungskontext bewegen. Dazu zählen Beispiele für Implementationen von IoT-Szenarien und dabei verwendete Techniken sowie allgemeine Betrachtungen von gängigen Protokollen, Vergleichen, wie auch Leistungsanalysen und Arbeiten, die sich mit den Besonderheiten und Auswirkungen von mobilen Verbindungen beschäftigen. Diese Ergebnisse werden genutzt, um im Weiteren eine funktionelle Einschätzung bestehender Protokolle zu erstellen. Mit Hilfe dieser Einschätzung erfolgt für eine engere Betrachtung eine Auswahl von geeigneten Protokollen. Protokolle zur Übertragung von periodischen Daten sind in der Regel Protokolle der Anwendungsschicht (vgl. Kapitel *). Das heißt, sie basieren auf Protokolle darunterliegender Schichten. Aus der Transportschicht finden wir hauptsächlich TCP und UDP. Diese werden in die Analyse einbezogen. Um die Kommunikation im Ganzen zu bewerten, implementieren die verschiedenen Schichten Dienstgüteparameter (*Quality of Service* oder *QoS*) in unterschiedlichem Umfang. Die Bedeutung der Dienstgüte bildet den Abschluss des Grundlagenkapitels

Es schließt sich der konzeptuelle Teil der Arbeit an. Dafür ist es notwendig, aus den vorrangegangenen Erkenntnissen eine konkrete Problemformulierung zu extrahieren und eine konkrete Zielstellung zu generieren. Die eigentliche Aufgabenstellung ist universell und lässt unterschiedliche Forschungsansätze zu. Im ersten analytischen Teil werden Ansätze besprochen und auf Probleme hingewiesen. Am Ende des Prozesses steht das Konzept einer Testumgebung für bestimmte Protokolle. Gestützt wird dieses Konzept durch das ACDS-Projekt (*), das als konkretes Szenario realistische Daten bereitstellt. Herausforderungen und Lösungsansätze werden erläutert. Ein zweiter Bereich orientiert sich an den Daten selbst. Dazu erfolgt eine gründliche Untersuchung der eingangs erwähnten Szenarien hinsichtlich der Daten. Es zeigt sich, dass periodische Daten Besonderheiten aufweisen, die es erlauben, die Übertragung in Daten und Struktur zu trennen sowie Optimierungen im Bereich der Datentypen vorzunehmen. Beides ermöglicht eine Reduzierung des Datenvolumens.

Das darauf folgende Kapitel beschreibt die Implementierung des Konzeptes. Dabei werden Details erläutert und auftretende Probleme besprochen. Sofern notwendig erfolgt eine Änderung des Konzeptes. Es wird beschrieben, wie die Zeitsynchronisation unterschiedlicher Rechner und eine automatisierte Testumgebung mit Hilfe von SSH-Befehlen realisiert sind. Ferner geht es um die Test selbst. Dazu mussten die Protokolle *MQTT* und *CoAP* angepasst werden.

Die eigentlichen Testergebnisse sind im Evaluationskapitel untergebracht. Hinzu kommen deren Interpretationen und daraus resultierende Vorschläge für konkrete Optimierungen. Angenommen wird, dass sich TCP innerhalb mobiler Kommunikation kritisch verhält und UDP die bessere Variante sein dürfte. Zu Klären ist die Frage, wie sich die unterschiedlichen Topologien, die sich aufbauend auf das jeweilige Protokoll ergeben, in einem realistischen Szenario verhalten. In die Betrachtung fließt auch ein, welche Auswirkungen unterschiedliche QoS-Parameter mit sich bringen.

Im letzten Kapitel erfolgt die Zusammenfassung der gewonnen Erkenntnisse sowie eine Bewertung von Konzept und Implementation. Erweitert werden die Ausführungen durch Vorschläge für Verbesserungen für folgende Arbeiten.

2 GRUNDLAGEN

2.1 KRITERIEN

Physikalische Topologie

In Computernetzwerken verbirgt sich hinter dem Topologie-Begriff die Anordnung von verbundenen Geräten. Auf Netzwerkebene beschreibt diese in der Regel die physikalischen Verbindungen und wird physikalische Topologie genannt. Zu den wichtigsten Grundtopologien gehören [BiBi02]:

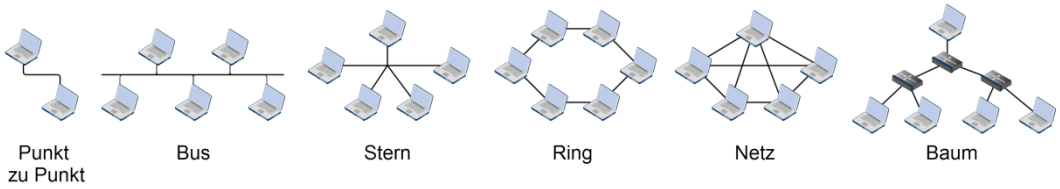


Tabelle 2.1: Physikalische Netztopogien.

Punkt-zu-Punkt

Die einfachste Topologie ist eine permanente Verbindung zwischen zwei Endpunkten. Der Vorteil einer solchen Verbindung ist eine ungestörte Kommunikation. In leitungs- und paketvermittelnden Netzen können Punkt zu Punkt-Verbindungen dynamisch auf- und abgebaut werden.

Bus

Busverbindungen finden sich in lokalen Netzwerken. Alle Geräte sind über ein Kabel miteinander verbunden. Nachrichten werden immer an alle angeschlossenen Teilnehmer versendet. Mit steigender Anzahl von Knoten steigt der Aufwand für die Netzwerkverwaltung. Das gemeinsame Kabel ist ein Single Point of Failure. Bei Problemen fällt das gesamte Netzwerk aus.

Stern

Häufig sind Topologien oder Teiltopologien als Stern umgesetzt. Dabei gibt es einen zentralen Verbindungspunkt, oftmals ein Netzwerk-Hub, -Switch oder -Router. Im Vergleich zur Bustoplogie werden mehrere Kabel benötigt. Dafür beeinträchtigt eine Kabelstörung lediglich ein Gerät und dessen Verbindung. Probleme im zentralen Hub betreffen hingegen alle angeschlossenen Geräte.

Ring

In einem Netzwerkring besitzt jeder Teilnehmer genau zwei Nachbarn. Alle Nachrichten wandern in einem Ring durch das Netzwerk und in gleicher Richtung. Alle Knoten agieren als Repeater. Eine Störung in einem Kabel kann die gesamte Kommunikation beeinträchtigen.

(Vermaschtes) Netz

Innerhalb vermaschten Netzen existieren sogenannte Routen. Im Gegensatz zu den bisherigen Topologien können Nachrichten auf dem Weg zu einem bestimmten Ziel verschiedene Routen/Pfade durch das Netzwerk nehmen. Ein Netzwerk, indem jeder Knoten mit jedem anderen Knoten verbunden ist, wird vollvermaschtes Netz genannt.

Baum

Eine Baumstruktur kombiniert Charakteristiken von Netz- und Sterntopologien. Sie besteht aus Gruppen von Sternnetzen, welche hierarchisch miteinander verbunden sind. Bäume erlauben die Erweiterung von bestehenden Netzwerken. Die Schwachstelle ist der Wurzelknoten des Baums. Bei einem Ausfall ist der komplette Unterbaum nicht erreichbar.

Hybrid

Hybride Netzwerke sind immer eine Kombination von zwei oder mehreren Grundtopologien. Das daraus resultierende Netzwerk ist keine Grundtopologie. Typische Beispiele sind Stern-Ring-Netzwerke oder Stern-Bus-Netzwerke.

Daisy Chain

Eine Daisy Chain ist die Verkettung von Punkt-zu-Punkt-Verbindungen. Bildlich gesehen kann die Kette als Ringtopologie verstanden werden, bei der eine Verbindung getrennt wurde. Häufig gibt es in einer solchen Kette einen Hauptrechner, der mit einer höheren Topologieebene verbunden ist. Die Knoten innerhalb einer Kette müssen für beide Richtungen als Repeater* fungieren können.

Logische Topologie

Neben der physikalischen Topologie existiert die logische Topologie. Sie beschreibt den Nachrichtenweg beziehungsweise den Datenfluss zwischen einzelnen Rechnern. Eine logische Topologie kann sich von der physikalischen unterscheiden. Logische Topologien sind an Netzwerk- und höhere Protokolle gebunden. Diese erzeugen sogenannte Overlaynetze, die eine von der physikalischen Schicht unabhängige Topologie erzeugen. Die Topologien sind identisch, mit dem Unterschied, dass selten Ring- oder Daisy-Chain-Strukturen vorkommen.

Stern- und Baumstrukturen gehören zu den zentralisierten Topologien, bei denen die Kommunikation immer über eine zentrale Vermittlerinstanz (einen Server) stattfindet. Alle Teilnehmer verbinden sich mit einem Server. Nachrichten werden von Teilnehmern an Servern und von Servern an Teilnehmer verschickt. Server können untereinander Nachrichten austauschen.

Vermaschte Netze und Punkt-zu-Punkt-Verbindungen gehören zu den dezentralisierten Strukturen, bei denen eine Nachricht direkt zwischen Sender und Empfänger vermittelt wird. Oft existieren redundante Wege für die Übermittlung einer Nachricht. Für eine funktionierende Kommunikation ist keine zentrale Instanz notwendig.

Peer-to-Peer

Auf höheren Ebenen erscheint hin und wieder der Begriff *Peer-to-Peer (P2P)*. Dieser beschreibt ein kommunizierendes Netz von gleichen Knoten. Gleich bedeutet im hiesigen Kontext, dass gleiche Dienste beansprucht und angeboten werden. Peer-to-Peer beleuchtet jedoch weniger die Topologie, sondern vielmehr die Anwendung und die Rollen, die die Knoten in dieser Anwendung einnehmen. Peer-to-Peer-Netze sind Overlaystrukturen der Anwendungsschicht. Diese Strukturen unterscheiden sich in *reine* P2P-Netze, in denen jeder Knoten Dienste in Anspruch nehmen und zur Verfügung stellen kann. In *zentralisierten* P2P-Netzen gibt es eine notwendige zentrale Instanz zur Verwaltung von anwendungsspezifischen Funktionen. Die dritte Form sind *hybride, hierarchische* P2P-Netze, in denen dynamisch Knoten als zentrale Server bestimmt werden können.

In der Telekommunikation beschreibt ein Nachrichtmuster das von einem Protokoll verwendete Nachrichtenschema, um einen Kommunikationskanal aufzubauen und zu nutzen.

Nachrichtenmuster

Push

Push ist eine Art der Kommunikation, bei der der Informationsfluss vom Sender (Informationsträger) gesteuert wird.

Pull

Im Gegensatz zu Push wird eine Pull-gesteuerte Übertragung vom Empfänger (Informationsbezieher) initiiert.

Long Polling

Long Polling Ist eine Modifikation der Pull-Übertragung, mit der eine Push-Übertragung emuliert werden kann. Ist die angefragte Information nicht sofort verfügbar, wird anstatt eine leere Antwort zu verschicken so lang gewartet, bis die Information verfügbar ist. Anschließend wird die Information an den Anfragenden übermittelt. Die Kommunikationskanäle bleiben gewöhnlich während dieses Zeitraums offen. *

One-Way/Fire and Forget

Fire and Forget ist das Verschicken einer Nachricht, ohne eine Antwort zu erwarten (Push). Ein Beispiel ist die Mitteilung über das Ende einer Operation, ohne eine Bestätigung dieser Mitteilungen zu erhalten. UDP ist ein klassisches One-Way-Protokoll. Radiostationen senden nach diesem Prinzip.

Request/Response oder Request/Reply

Das Anfrage/Antwort-Modell charakterisiert die Basiskommunikation zwischen zwei Rechnern. Rechner A sendet eine Anfrage nach bestimmten Daten und Rechner B antwortet entsprechend (Pull), ähnlich wie bei einem Telefongespräch. Es ist ein einfaches, weit verbreitetes, synchrones Muster, insbesondere in Client-Server-Architekturen.

Nachrichten-Queues

Die Kommunikation mit Hilfe von Queues ist asynchron. Sender und Empfänger müssen nicht zur gleichen Zeit mit der Nachrichten-Queue interagieren. Eine Queue speichert empfangene Nachrichten, bis sie ausgeliefert oder abgeholt werden (Push oder Pull).

Publish/Subscribe

In Publish/Subscribe-Strukturen werden Nachrichten nicht direkt vom Sender an Empfänger verschickt. Stattdessen werden klassenähnliche Strukturen eingeführt, um die Nachrichten zu charakterisieren. Sender besitzen kein Wissen darüber, wer die Empfänger sind oder wie viele existieren. Gleiches gilt für die Empfängerseite, welche keine Kenntnis über den Sender hat. Pusblish/Subscribe ist eine Variante einer Queue (Push).

Routing-Schema

Ein Routing-Schema in der Netzwerkkommunikation beschreibt die Verteilung und Steuerung von Nachrichten. Die fünf bekanntesten Schemata sind Unicast, Broadcast, Multicast, Anycast und Geocast.

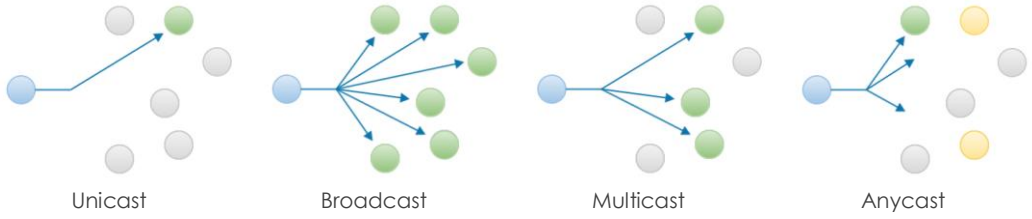


Tabelle 2.2: Routing Schemata in der Netzwerkkommunikation.

Unicast

Bei einer Unicast-Kommunikation gibt es genau einen Sender und einen Empfänger. Unicast ist das dominierende Schema im Internet.

Broadcast

Ein Broadcast ist das Verschicken einer Nachricht an alle im Nachrichtennetz befindlichen Teilnehmer.

Multicast

Ein Multicast ist ein eingeschränkter Broadcast, bei dem eine Nachricht an eine Teilmenge von Teilnehmern versendet wird.

Anycast

Bei einem Anycast werden mehrere Teilnehmer über dieselbe Adresse angesprochen. In der Regel antwortet derjenige, der über die kürzeste Route erreichbar ist.

Geocast

Geocast ist eine spezielle Form des Multicast. Die Teilmenge an Empfänger wird durch ein räumlich abgegrenztes Gebiet bestimmt.

Auslieferungsgarantien

Nachrichten können auf dem Weg durchs Netz mit bestimmten Sicherheiten übertragen werden. Den Grad der Sicherheit beschreiben die folgenden Garantien für eine Auslieferung.

Best effort

Best effort definiert die niedrigste Sicherungsbene. Es gibt keine Garantie für eine Auslieferung. Diese wird so gut und so schnell wie möglich betrieben, in Abhängigkeit der zur Verfügung stehenden Ressourcen. Es werden keine verlorenen oder doppelten Pakete erkannt.

At most once

At most once ist eine best effort-Auslieferung, bei der doppelte Pakete erkannt und deren Auslieferung verhindert wird.

At least once

At least once garantiert die Auslieferung einer Nachricht, verhindert aber nicht, dass Nachrichten doppelt auftreten können.

Exactly once

Garantiert wie at least once die Auslieferung einer Nachricht und erkennt und behandelt doppelte Nachrichten bei der Übertragung

Garantierte Reihenfolge

Nachrichten können unterschiedliche Routen auf dem Weg zum Empfänger nehmen. Dadurch kann es vorkommen, dass eine Reihe von Nachrichten nicht in der Reihenfolge ankommen, wie sie versendet wurden. Einige Protokolle implementieren Mechanismen, um die ursprüngliche Reihenfolge wieder herzustellen.

Garantierte Datenintegrität

Eine Nachricht kann über das Netz nicht nur verloren gehen, sondern ihre eigentliche Information kann verändert werden, indem sich zum Beispiel einzelne Bits während der Übertragung ändern. Gleichermaßen gibt es hierfür Mechanismen zur Erkennung und Behandlung.

Eine Garantie ist immer *Ende-zu-Ende* über die gesamte Verbindung zu bewerten. Eine *Ende-zu-Ende*-Garantie ist die minimale Garantie aus allen bei einer Übertragung beteiligten Knoten. Garantieren 99 Knoten eine zuverlässige und dopplungsfreie Auslieferung, 1 Knoten jedoch eine Auslieferung nach best effort-Semantik, kann für die gesamte Verbindung nur eine best effort-Garantie erreicht werden.

2.2 ANWENDUNGSPROTOKOLLE

Protokolle der Anwendungsschicht stellen erweiterte Funktionen für Anwendungen bereit. Dazu zählt das Erstellen von Verbindungen zu unteren Schichten, ebenso das Verwalten von Sitzungsparametern und Aufbereitung von Anwendungsdaten.

2.2.1 XMPP

XMPP [Stpe00] steht für *Extensible Messaging and Presence Protocol* und hieß ursprünglich *Jabber* [00b]. Anfänglich für Instant Messaging genutzt, etablierte sich *XMPP* im Laufe zu einem gezielt erweiterbaren XML-Routing-Protokoll. XMPP besteht aus einem Protokollstabel im Kern (RFC 6120 [Sain11a], RFC 6121 [Sain11b], RFC 6122 [Sain11c]) und einer Reihe offizieller und experimenteller Erweiterungen, die von der *XMPP Standards Foundation* unter der Bezeichnung *XMPP Extension Protocols* entwickelt und verwaltet werden.

Neben Instant Messaging, der Übermittlung von Textnachrichten und Unterstützung von Gruppenchats werden Zustände und die Anwesenheit von Nutzern verarbeitet. Desweiteren stehen Mechanismen zur integrierten Dateiübertragung und Zertifikatsvermittlung bereit. XMPP kann über *Jingle* [LBSM09] zur Signalisierung und Verwaltung von XMPP getrennten unabhängigen Verbindungen verwendet werden.

XMPP ähnelt im Aufbau *SMTP*. Ein Nutzer wird durch eine E-Mail-ähnliche Adressstruktur identifiziert, wobei Geräte ebenfalls Nutzer repräsentieren können. Die Übertragung wird von Nutzerseite initiiert und ist mit einer zentralen Vermittlungsstelle Push-orientiert. Diese XMPP-Server können sich zu beliebigen Netzen zusammenschließen und ermöglichen Förderationsstrukturen.

2.2.2 MQTT

MQTT ist ein leichtgewichtiges Push-Protokoll zur Übertragung von binären Daten. *Message Queue Telemetry Transport* ist eine von *IBM* [10a] bereitgestellte offene Spezifikation, die seit 2013 eine Standardisierung durch *OASIS* [RaRG14] erfährt und für ressourcenarme Umgebungen mit unzuverlässigen Verbindungen und eingeschränkter Bandbreite geeignet ist. Daher wird *MQTT** als wichtiger Vertreter in M2M-Umgebungen, Sensornetzwerken, Handhelds und *Embedded Systems* gesehen. *Facebook Inc*. implementiert *MQTT* für den mobilen *Facebook Messenger* [00c].

Nutzer kommunizieren nicht direkt miteinander, sondern registrieren sich ganz im Sinne von Publish/Subscribe für ein Topic. *MQTT* arbeitet mit einem zentralen Broker, der als Vermittler zwischen Nutzern und Topics agiert. Topics werden durch Zeichenketten abgebildet, wobei hierarchische Baumstrukturen möglich sind. Die verschiedenen Ebenen werden durch ein „/“ getrennt. Ein Topic kann Knoten und Blatt zugleich sein. Es erfolgt innerhalb *MQTT* keine Unterscheidung zwischen Knoten und Blättern. Ein Nutzer kann sich allein für ein Topic oder über Wildcards für ein Topic und seine Untertopics registrieren. Nutzer werden über eine beliebig gewählte ID identifiziert, welche maximal 23 Zeichen lang ist und auf einem Server eindeutig ist. Erfolgt eine weitere Anmeldung mit identischer ID, wird die bestehende Verbindung verworfen.

MQTT unterstützt drei Auslieferungsmodelle: At most once, at least once und exactly once. Für die beiden letzteren erhält der Nutzer eine Bestätigung über die Auslieferung der Nachricht an alle Empfänger. *MQTT* implementiert dafür Möglichkeiten der temporären und persistenten Speicherung von Nachrichten auf Nutzer und Brokerseite. Hinzu kommen Mechanismen zum Erkennen von doppelten Nachrichten über Flags im Nachrichtenheader.

2.2.3 CoAP

Wie der Name andeutet, ist *CoAP* oder *Constrained Application Protocol* ein für beschränkte Anwendungen designtes Anwendungsprotokoll zur Datenübertragung. Daten werden per Anfrage ausgeliefert. CoAP ermöglicht daher vorwiegend eine Pull-orientierte Kommunikation. Dafür orientiert sich das Protokoll am REST-Architekturstil. Datensätze und Objekte werden innerhalb von CoAP durch URIs identifiziert. Für die Discovery von Ressourcen steht ein protokollinterner Mechanismus zur Verfügung. Durch die Orientierung an *RESTfull*-Prinzipien können CoAP-Strukturen mit wenig Aufwand an HTTP-Schnittstellen gebunden werden. Darüber hinaus ist CoAP so konstruiert, dass asynchrone Kommunikationsmuster bis hin zu Pusblish/Subscribe zu realisieren sind. Spezifikation und Standardisierung werden durch die IETF vorangetrieben und befinden sich im finalen Prozesszyklus.

Nachrichten können durch vorherige Konfiguration zuverlässig übertragen werden. Die Auslieferung erfolgt per Unicast oder Multicast. Asynchrone Übertragungen werden per Token realisiert. Ein Token repräsentiert einen anfragenden Nutzer. Auf beiden Seiten wird es notwendig, sich den Zustand der Kommunikation zu merken. Da als Transportprotokoll UDP verwendet wird, können Multicast-Anfragen effizient in IP-Netzen angewendet werden. Ferner können durch den Wechsel auf DTLS als Transportprotokoll Sicherheitsmechanismen implementiert werden.

2.2.4 AMQP

Advanced Message Queuing Protocol ist ein offener Standard zur Nachrichtenübertragung und ist für nachrichtenorientierte Middlewares konzipiert. *AMQP* stellt daher nicht die Middleware an sich, sondern ein mögliches Protokoll innerhalb einer MOM dar. Die Kernziele sind Offenheit, Sicherheit, Zuverlässigkeit und Interoperabilität. *AMQP* stellt Broker zur Vermittlung von Nachrichten bereit. Die Kommunikation selber ist Queue-orientiert. Ein Broker realisiert die technische Übertragung von Nachrichten zur entsprechenden Queue. Dabei können an der Empfängerseite eine oder mehrere Queues adressiert werden. Die Adressierung übernimmt ein flexibles Routingschema, das Semantiken, wie *point-to-point, fanout, publish/subscribe* und *request/response* zulässt.

Das Routing der Nachrichten selber wird der Anwendung überlassen. Die entsprechenden Informationen sind im Nachrichtenheader integriert, der wie ein adressierter Umschlag fungiert. Nachdem eine Nachricht in einer Queue eintrifft, wird sie je nach Konfiguration an einen Empfänger weitergeleitet oder der Empfänger fragt von sich aus die Queue nach neuen Nachrichten ab. Daher sind asynchrone und synchrone Szenarien möglich.

AMQP ist stark an *JMS (Java Messaging Service)* angelehnt und ermöglicht eine einfache Integration bestehender JMS-Implementationen. Daten werden binär übertragen. Die Standardisierung erfolgt durch OASIS [RaAn12]. Häufig wird AMQP in Cloud-Umgebungen eingesetzt, da von Anfang an Clustering, Förderation, Skalierung, Fehlertoleranz und Transaktionen innerhalb Multiplattformszenarien in die Spezifikation eingearbeitet sind.

2.2.5 DDS

OMG Data-Distribution Service [07], kurz *DDS*, hebt sich zu den bisherigen Techniken insofern ab, da die als Middleware konzipierte Software versucht, einen ganzheitlichen Ansatz von Netzwerk bis Datenebene zu vereinheitlichen. Grundsätzlich ist *DDS* ein Standard für Publish/Subscribe-Kommunikation innerhalb eingebetteter Systeme zur Datenverteilung in Echtzeit. Die Verteilung erfolgt über heterogenen datenzentrierten Systemen innerhalb der verlässlichen, skalierbaren und QoS-fähigen Informationsstruktur. DDS formt Datenströme per QoS und versucht vorhersagbare Ergebnisse zu liefern.

Kernfähigkeiten sind die automatische Integration neuer Anwendungen in Unabhängigkeit zur Netzwerktopologie, eine breit gefächerte Behandlung von QoS-Parametern, Redundanz und Fehlertoleranz, Interoperabilität und Entkopplung im System befindlicher Entitäten von Zeit, Raum und Synchronisation. DDS ist eine *low latency* Kommunikationsarchitektur mit abstrakter Datenebene. Dadurch müssen sich Anwendungen nicht um die Datenrepräsentation, Prozessorarchitektur, Betriebssystem oder Programmiersprache auf der Gegenseite kümmern.

DDS implementiert eine datenzentrierte Publish/Subscribe Kommunikation. Daten werden im Netzwerk durch einen Datenbus repräsentiert, der Domänen, Topics und Instanzen verwaltet. Die Übertragung wird durch ein eigens entwickeltes auf UDP basierendes Protokoll namens RTPS (Real Time Publish Subsribe [10b]) arrangiert. Die Verteilung erfolgt dezentral und kann ohne Vermittler direkt zwischen zwei Teilnehmern erfolgen. Daher ergeben sich Peer to Peer Strukturen mit Multicastanforderungen, die in IP-Netzwerken effizient umgesetzt werden können.

2.2.6 STOMP

STOMP steht für *Simple Text Oriented Message* Protocol und wurde vorher *TTMP genannt*. *STOMP* tauscht Textnachrichten aus und wird von einer Vielzahl von MoMs unterstützt *. Durch die Spezifikation als Wire Protocol können Clients und Server unabhängig von der Programmiersprache miteinander kommunizieren. *STOMP* benötigt eine Vermittlerkomponente als Nachrichtenverteiler. Die Vermittlung erfolgt über Publish-Subsribe, wobei Teilnehmer sich für eine *Destination* anmelden und Nachrichten an eine *Destination* adressieren.

Die Nachrichtensemantiken ähneln denen von HTTP. Auf der Transportebene kommt TCP zum Einsatz. Kommuniziert wird über Frames, welche die Textnachrichten kapseln. Die erste Zeile enthält den Befehl, darauf folgt der Header, danach der Inhalt und das Ende der Nachricht markiert ein Null-Zeichen.

2.2.7 WEB-ORIENTIERTE PROTOKOLLE

Die folgenden Protokolle teilen die Eigenschaft, dass sie für Browserumgebungen entwickelt wurden.

OData

OData ist ein offenes Protokoll, das von Microsoft im Rahmen der *Open Data*-Bewegung entstand und innerhalb der *Open Specification Promise* veröffentlicht wurde [00d]. Mittlerweile existiert Version 4.0, die innerhalb von *OASIS* standardisiert wird [BaRa14]. Der Zugriff auf Daten erfolgt Pull-basiert über gesteuerte Anfragen.

Die Intention hinter *OData* ist die Schaffung einer allgemeinen Schnittstelle für eine Vielzahl von verteilten Daten, ganz im Sinne von *Open Data*. Mittlerweile gibt es Webdienste für *ebay, Mircosofts Sharpoint, Stack Overflow*, innerhalb *SAP* und *IBM*-Anwendungen.

Basis von *OData* ist *AtomPub*. Daten werden per JSON übertragen. In der Regel handelt es sich um Feeds. Die Referenzierung ist per URI realisiert. Daneben erlauben Metadaten die Beschreibung von Strukturen und Ressourcen, sodass eine Trennung von Daten und deren Beschreibung über Metadatendokumente möglich ist. Alternativ werden große Datenpakete, hauptsächlich Dateien in getrennten binären Strömen, versendet. Die Ausführungen gelten gleichermaßen für *GData* [12], ein ähnliches von *Google* entwickeltes Protokoll.

AtomPub

Wie bereits erwähnt, nutzen *OData* und *GData* zur Übertragung *AtomPub* [GrHo07]. *AtomPub* basiert wiederum auf *HTTP* und wurde zur Bearbeitung von XML- und HTML-Ressourcen entwickelt. Der Kern ist als REST-Schnittstelle implementiert und bietet die üblichen GRUD-Operationen. *AtomPub* ist nicht mit dem Begriff *Atom* gleich zu setzen, der zum Teil auftaucht, um das Protokoll zu benennen. *Atom* ist ein Überbegriff, unter dem sich des Weiteren das *Atom Syndication Format* verbirgt [NoSa05]. *AtomPub* ist ein *IETF*-Standard und überträgt Daten wie Text, HTML, XML, Base64, ebenso Videos und Audios. Die Kommunikation ist Pull-gesteuert. Die gesamte HTTP-Kommunikation geht vom Nutzer aus.

Websockets

Die Entwicklung von *Websockets* zielte darauf ab, die immer deutlicheren Nachteile der globalen Client-Server-Kommunikation im Internet entgegen zu wirken und den entsprechenden Anforderungen einer zukünftigen allgegenwärtigen Vernetzung gerecht zu werden. Insbesondere ging es darum, die Client-Server-Anwendungen innerhalb gängiger Webbrowser und -server zu optimieren, die in der Regel auf *HTTP* bauten, sodass die Kommunikationssteuerung vom Client ausging und eine Aktualisierung durch ständiges Polling angefragt werden musste. *Websockets* bieten nun full-duplex-fähige Kanäle über eine einzige TCP-Verbindung, die es auf der Serverseite ermöglichen, Nachrichten unaufgefordert an den Browser zu senden. Somit wird eine push-gesteuerte Kommunikation möglich. *Websockets* sind an sich kein eigenständiges Protokoll, sondern eher als API zu verstehen, die Sockets für Browseranwendungen zur Verfügung stellt. Die Standardisierung lieg in den Händen der IETF [FeMe11].

PubSubHubbub

Hinter dem etwas sperrigen Begriff *PubSubHubbub* verbirgt sich ein *Webhook*-basiertes Publish/Subscribe-Protokoll zur Bereitstellung von Ressourcen im Web. Im Grunde funktioniert die Kommunikation wie bei einem gängigen HTTP-Request. Ergänzend dazu schickt die anfragende Seite eine URL. Diese dient später dazu, die anfragende Seite über Aktualisierungen zu informieren. Daraus ergibt sich ein Server-zu-Server-Modell, da alle Instanzen über öffentliche URLs erreichbar sein müssen. Daten werden im System als Ressourcen abgelegt, die als Topics bezeichnet werden und eine URL erhalten. Clients können sich gemäß Publish/Subscribe für solche Topics registrieren. Bei Aktualisierungen wird nicht direkt der neue Datensatz verteilt, sondern registrierte Instanzen werden über die Aktualisierung informiert, woraufhin diese explizite Datenanforderungen abschicken können.

2.2.8 ØMQ

ZeroMQ, 0MQ, ZMQ und ØMQ sind Begriffe für dieselbe Messaging-Bibliothek. Bei *ØMQ* handelt es sich um eine hoch performante, asynchrone API für skalierbare, verteilte oder konkurrierende Anwendungen. Es ist daher kein Protokoll im bisherigen Sinne. Vielmehr ist *ØMQ* für Entwickler gedacht, die eine flexible, leichtgewichtige und schnelle Technik suchen, um Nachrichten im System zu verteilen. Dafür stehen Nachrichtenqueues bereit. Es gibt jedoch keinen notwendigen zentralen Broker. Es können verschiedene Topologien aufgebaut werden, Request-Reply, Publish/Subscribe, Push/Pull oder Exclusive Pair. Neben der API existiert das *ZeroMQ Message Transport Protocol*, welches innerhalb der *Digital Standards Organization* standardisiert wurde. *ØMQ* platziert sich auf der Netzwerkebene. Viele Anwendungsfunktionalitäten müssen hinzu programmiert werden.

2.3 TRANSPORTPROTOKOLLE

Aufgaben der Transportschicht sind die Segmentierung und Steuerung des Datenstroms sowie die Behandlung von Übertragungskonflikten. Transportprotokolle erstellen Ende zu Ende-Verbindungen und vereinheitlichen die Übertragung für darüber liegende Schichten.

2.3.1 TCP

Das *Transmission Control Protocol* wird zur Übertragung von Daten zwischen Rechnern verwendet. Es ist als zuverlässiges, verbindungsorientiertes, paketvermittelndes Transportprotokoll in *RFC 793* spezifiziert [Post00a]. *TCP* stellt vor dem Datentransport eine Vollduplex-Verbindung zwischen zwei Endpunkten her, oftmals Sockets genannt. Die Übertragung erfolgt in beide Richtungen. Über die Bestätigung von Datenpaketen wird ein Verlust eben solcher erkannt und darüber hinaus eine Netzüberlastung verhindert.

Daten werden von *TCP* segmentiert übertragen. Die maximale Größe eines Segments kann bis zu 1500 Byte sein, wird jedoch von der darunter liegenden Netzwerkschicht und damit in der Regel von *IP* bestimmt. Dadurch schwanken die Segmentgrößen zwischen 64 bis 1500 Bytes. *TCP* implementiert eine Fluss- und Überlastungssteuerung und passt das übertragene Datenvolumen dynamisch an.

Eine TCP-Verbindung gliedert sich in die drei Phasen Initialisierungsphase, Phase der Nutzdatenübertragung und Verbindungsabbauphase. Die genannten Eigenschaften machen *TCP* zu einem weniger leichtgewichtigen Protokoll mit komplexerer Transportsteuerung und dafür notwendigen Overhead.

2.3.2 UDP

Das *User Datagram Protocol* ist ein leichtgewichtiges, verbindungsloses Protokoll der Transportschicht zur Übertragung von Daten im Internet. Verbindungen sind nicht zuverlässig. Pakete können verloren gehen. Es können Dopplungen auftreten und es wird nicht garantiert, dass die Reihenfolge der empfangenen Pakete, die der gesendeten entspricht. Ferner erfolgt die Übertragung ungeschützt.

UDP wurde als Alternative zu *TCP* entwickelt, um Sprachdateien verzögerungsfrei übertragen zu können. Da kein Verbindungsaufbau nötig ist, bietet sich *UDP* besonders für kleine Datenmengen an. Die maximale Größe eines UDP-Datagramms beträgt 65.535 Bytes, wobei dieses auf Netzwerkebene, zum Beispiel durch IP, fragmentiert übertragen wird. Der Protokollheader ist simpel*, der Overhead dadurch gering und UDP-Pakete können mit wenig Aufwand verarbeitet werden.

UDP wird heute vorwiegend zur Übertragung von Audio- und Videodateien, zur Service Discovery, beispielsweise *DNS*, und für Broadcastanfragen verwendet, da es als verbindungsloses Protokoll IP-Multicast unterstützt. *UDP* wird durch *RFC 768* spezifiziert [Post00b].

2.3.3 STCP

Das *Stream Control Transmission Protocol* ist wie *TCP* und *UDP* ein Protokoll der Transportschicht. Es arbeitet damit unmittelbar über der Netzwerkschicht. *SCTP* gilt als zuverlässiges, verbindungsorientiertes Protokoll, wobei viele Eigenschaften optional zuschaltbar sind. *SCTP* versucht damit die Nachteile von *UDP* und *TCP* zu lösen. Es ist ein relativ junges Protokoll und in *RFC 3286* spezifiziert [Ongl02].

SCTP ist gleichermaßen ein nachrichtenorientiertes Protokoll, das Pakete auf Wunsch zuverlässig und mit garantierter Paketreihenfolge versendet. SCTP unterstützt *Multi-Streaming* und *Multi-Homing*. *Multi-Streaming* ermöglicht die logische Trennung von mehreren Datenströmen innerhalb einer Ende-zu-Ende-

Verbindung mit einer voneinander unabhängigen Transportsteuerung. *Multi-Homing* ist die Adressierung von Endpunkten durch mehr als eine IP-Adresse. Diese Redundanz erhöht die Ausfallsicherheit im Falle eines Failovers.

Ursprünglicher Anwendungsfall sind Telefonverbindungen über das Internet, bei denen Signalinformationen gleichzeitig mit Sprachdaten übertragen werden. *SCTP* wird durch mangelnde Unterstützung noch selten eingesetzt. Ebenso gibt es bisher kaum experimentelle Erfahrungen. Wohl scheinen Probleme innerhalb *NATs* aufzutreten *.

2.4 NETZWERKSCHICHT

Die Netzwerkschicht umfasst im Folgenden alle Schichten unterhalb der Transportschicht. Im Fokus liegen der Netzwerkzugang und Einflüsse auf die Übertragungsqualität.

2.4.1 KERNNETZWERK

Unter dem Begriff Kernnetz ist der Verbund von verschiedenen Netzwerkschnittstellen, Netzwerkbetreibern und Netzwerkdomänen zu verstehen. Charakteristisch hierfür sind transportbestimmende Funktionalitäten in Kombination mit Routern, Switches und Firewalls. Wie der Begriff *Internet* bereits andeutet, findet der Großteil der computergestützten Kommunikation nicht ausschließlich in abgeschotteten Netzen statt*. Verschiedenste Technologien können auf dem Weg einer Nachricht zum Einsatz kommen. Sowohl die Technologie selbst, wie auch deren Schnittstellen beeinflussen die Kommunikation.

2.4.2 IP-NETZWERKE

Im Rahmen der Arbeit wird sich hauptsächlich auf IP-basierte Systeme beschränkt. Grund dafür sind die weite Verbreitung und Unterstützung des IP-Protokolls im globalen Netz sowie die wachsende Bedeutung von *ALL-IP-Netzen (AIPN)*. In *AIPNs* sind verschiedenste Dienste von überall zu jeder Zeit erreichbar und die Platzierung neuer Dienste ist ohne Weiteres im IP-Netzwerk möglich. ALL-IP-Netze sind eine Realisierung eines *Next-Generation Networks (NGN)*. Des Weiteren konzentriert sich die Betrachtung auf den im Internet etablierten *TCP/IP-Protokollstack*, der gelegentlich unter der Bezeichnung *Internetprotokollfamilie* auftaucht. Es handelt sich um eine Sammlung von Kommunikationsprotokollen, die auf IP aufbauen. In IP-Netzen werden Daten in Paketen und nach „bestem Bemühen“ (best effort) übertragen, sprich unzuverlässig, zustandslos und ohne Fehlerbehandlung. Das IP-Protokoll ist hauptsächlich für die Adressierung und das Routing konzipiert. Jedes IP-Paket enthält eine Ziel- und Quelladresse. Auf dem Weg durchs Netzwerk werden diese Informationen genutzt, um das Paket erfolgreich beim Empfänger abzuliefern. Mehrere Pakete mit gleicher Zieladresse können unterschiedliche Routen durch das Netzwerk nehmen. Es ist den höheren Schichten überlassen, die Zuverlässigkeit der Datenübertragung zu gewährleisten. Zurzeit gibt es zwei aktive Varianten des Protokolls: IPv4 und dessen Nachfolger IPv6.

2.4.3 UNTERE SCHICHTEN

Die Schicht unterhalb der IP-Ebene wird maßgeblich vom physikalischen Übertragungsmedium bestimmt. Dabei kann grob zwischen drahtgebundener und drahtloser Übertragungstechnik unterschieden werden. Zu ersterer gehören *Ethernet* oder *Token Ring*. Zu letzterer WLAN (IEEE-802.11) oder Mobilfunkstandards wie GPRS, UMTS oder LTE. Die drahtgebundene Übertragung zeichnet sich durch hohe Geschwindigkeiten, hohe Ausfallsicherheit und stabile Übertragungsbedingungen aus. Daten werden oft unzuverlässig versendet, da Paketverluste selten auftreten. Anders verhält es sich mit drahtlosen Techniken. Die Übertragung ist fehleranfälliger. Schwankungen der Eigenschaften sind typisch. Es treten häufiger Paketverluste auf und es kann zu unvorhersehbaren Verbindungsabbrüchen kommen. Daher gibt es eine Vielzahl von Mechanismen, welche eine sichere drahtlose Übertragung gewährleisten sollen. Diese werden unter anderem im Abschnitt *Quality of Service* vorgestellt. Zuvor gibt es einen Überblick über die Besonderheiten mobiler Übertragung.

2.4.4 MOBILE VERBINDUNGEN

Bei einem Handover (Verbindungsübergabe) wechselt ein mobiles Endgerät während einer Funkverbindung die Funkzelle beziehungsweise den Zugangspunkt im Funknetz. Die Verbindung muss dabei nicht zwingend unterbrochen werden. Es gibt eine Reihe verschiedener und redundanter Begriffe für die nähere Charakterisierung.

Smooth/Seamless Handover

Bei einem Handover wird in der Regel eine Übertragungssitzung abgebrochen. Das führt dazu, dass der Datenfluss unvollständig übertragen wird und oft die Übertragung von vorn beginnt. Insbesondere ein vertikaler Handover kann zum Bezug einer neuen IP-Adresse führen. Ein SH versteckt den Übertragungsabbruch und nachdem eine neue Verbindung besteht, wird die kurzzeitig unterbrochene Verbindung weitergeführt.

Hard Handover

Wird bei einem Handover zuerst eine bestehende Verbindung vollständig abgebaut, bevor die Verbindung zur neuen Zelle hergestellt wird, spricht man von einem Hard Handover.

Soft Handover

Dagegen existiert bei einem Soft Handover die neue Verbindung bereits vor dem Abbau der alten Verbindung. So existieren während eines kurzen Zeitabschnitts zwei Verbindungen zu verschiedenen Zellen.

Vertikaler Handover

Ein vertikaler Handover verweist auf einen Netzwerkknoten, der seinen Verbindungstyp ändert, um Zugang zu einer Infrastruktur zu bekommen und Knotenmobilität zu gewährleisten. Anders gesagt: Ein vertikaler Handover ist ein automatischer Übergang von einer Technologie zu einer anderen und damit eine Änderung auf Sicherungsschicht. Dazu zählt die Verbindungsübergabe von GSM/UMTS zu WLAN. Die Verbindung muss dabei in der Regel ab- und neu aufgebaut werden, da sich die IP-Adressen ändern.

Horizontaler Handover

Ein horizontaler Handover passiert stattdessen auf der gleichen Netzwerkebene, sprich ohne Änderung der Netzwerkschnittstelle, zum Beispiel zwischen zwei WLAN Zugangspunkten. Je nach Technologie- und Netzwerkkonfiguration können Verbindungsunterbrechungen verhindert werden.

Media-independent Handover

Ein media-independet Handover ist ein IEEE 802.21-Standard, um einen Handover von einer Technologie der zweiten Schicht zu einer anderen zu gewährleisten, ohne bestehende IP-Sitzungen zu beenden.

2.5 QUALITY OF SERVICE

Quality of Service (QoS), zu Deutsch *Dienstgüte* ist eine Menge von Dienstanforderungen, welche während eines Transportflusses bereitgestellt werden (müssen). Sie ermöglichen eine bessere Verwaltung von Ressourcen im Netzwerk und steigern darüber hinaus die Effizienz der Komponenten. *QoS*-Regeln erlauben den Anwendungen Datenflüsse in einem Netzwerk zu verwalten, zu priorisieren und zu formen. Die Anforderungen an das Netzwerk sind durch Dienstanforderungen bestimmt, welche durch die Endnutzeranwendung spezifiziert werden. Unter der Bereitstellung von Dienstgüte versteht man eine Menge an Techniken, um auf Latenzen, Streuungen und Stauereignisse innerhalb eines Netzwerks einzuwirken. Eine andere Definition ist: „Menge von quantitativen und qualitativen Charakteristiken eines verteilten Multimedia-Systems, welche notwendig sind, um eine geforderte Funktionalität einer Anwendung zu erreichen“ [VKBG95]. Quality of Service ist die Sicht des Anbieters auf einen Dienst oder des Betreibers auf ein Netzwerk.

QoS sind eng verwandt mit *Quality of Experience (QoE)*. *QoE* ist die Bewertung eines Dienstes aus Nutzersicht. Hierbei werden subjektive Eindrücke mit eingebunden. Es wird der gesamte Kommunikationsprozess bewertet und umfasst die gesamte Ende-zu-Ende-Vermittlung, ohne sich auf das Netzwerk zu beschränken. Typische Evaluationsmetriken sind Videoqualität und Energieeffizienz. QoE ist insbesondere für die Sicherstellung der Kundenzufriedenheit wichtig. In der vorliegenden Arbeit werden QoE-Aspekte nicht näher beleuchtet.

Qualitätsparameter sind schichtweise definiert. Die Zuordnungen von Funktionen zu Schichten sind nicht immer eindeutig. Zum Beispiel arbeiten Router, die auf Paketinformationen, wie Protokoll oder Port zugreifen auf Schicht 3 und 4. Die Betrachtung beschränkt sich im Grundlagenkapitel auf die Transportebene und darunter liegende Schichten. Eine Untersuchung der QoS-Fähigkeit der vorgestellten Anwendungsprotokolle erfolgt im Analysekapitel.

2.5.1 METHODEN

Ausgangspunkt ist eine Übertragung von Daten in IP-Netzen, die bestimmte Qualitätsanforderungen genügen soll. Es werden grundsätzlich zwei Kategorien unterschieden: präventive und reaktive Verfahren.

Präventive Verfahren

Ziel von präventiven Verfahren ist ein bestimmtes Maß an Qualität zu erreichen und beizubehalten. Grundlage ist die Überwachung von Verkehrsparametern

(*Traffic Policing*). Hinzu kommen Rufannahmesteuerung, Verkehrsformung (*Traffic Shaping*) und Scheduling-Verfahren.

Tabelle 2.3: Präventive Verfahren zur Bereitstellung von QoS.

Bei der Verkehrsformung wird die Größe der Pakete verändert und gegebenfalls mehrere Pakete gebündelt. Damit sollen Stauungen in den Verbindungen verhindert werden. TCP arbeitet mit solch einem Mechanismus. Durch *Scheduling* wird bestimmt, wie mit der Weiterleitung von Paketen in Warteschlangen verfahren wird. Die einfachste Variante arbeitet nach dem Prinzip: kommt zuerst, geht zuerst (*FIFO*). Anspruchsvollere Ansätze priorisieren die einzelnen Pakete, um die Verarbeitung wichtiger Pakete zu beschleunigen. Für eine faire Behandlung aller Pakete eignen sich Erweiterungen wie *Weighted-Fair-Queuing*. Ein vergleichsweise primitiver Ansatz ist *Overprovisioning*. Dabei werden überdurchschnittlich viele Ressourcen bereitgestellt, um Peak-Ereignisse behandeln zu können. Dies bietet sich für Netzwerke mit vorhersehbarer Verkehrslast an. QoS-Garantien können nur als Wahrscheinlichkeit angegeben werden. Die geringe Eignung zeigt sich an TCP. Durch die schrittweise Erhöhung des Netzwerkverkehrs stößt man unmittelbar an die Grenzen von *Overprovisioning*.

Reaktive Verfahren

Zu den reaktiven Verfahren gehören Flusskontrollverfahren und das selektive Löschen von Paketen (*Early Packet Discard*). Dazu zählen *ABR* - eine ratenbasierte Flusssteuerung mit expliziter Steuerung*. TCP mit seiner fensterbasierten Flusssteuerung ist eine implizite Steuerung. Der Fokus liegt in der Vermeidung einer Überlastung in Knoten mit geringer Datenrate. Hinzu kommen Mechanismen zur Behandlung von Stauungen.

2.5.2 ARCHITEKTUREN

Für IP-Netze gibt es eine Reihe von Modellen für die Steuerung von Verkehrsströmen in paketorientierten Netzen. Die wichtigsten sind in Tabelle 2.4 zusammengefasst.

Bezeichnung	Beschreibung
Best Effort	Es gibt keine Garantie für die Auslieferung von Paketen. Das Paket wird so schnell wie möglich mit den zur Verfügung stehenden Ressourcen übertragen.
intServ/RSVP (Resource reSerVation Protocol)	Es gibt eine garantierte Dienstgüte durch Reservierung von Ressourcen im Netzwerk. *RSVP* ist Signalisierungsprotokoll um Knoten entlang eines Pfades und für Ströme zu konfigurieren. Zustände müssen über gesamten Pfad konsistent sein und in Routern gehalten werden. RSVP erzeugt viele UDP-Nachrichten. *IntServ* skaliert schlecht und ist komplex für große Netzwerke.
diffServ	Verhindert Zustandsinformationen an Knoten und führt Klassifizierung von Paketen ein. Damit wird Skalierbarkeit erhöht. Ströme mit gleicher Klassifizierung können aggregiert werden. IP-Field im Header definiert Per hop behavior (PHB), wonach Router entsprechend reagieren.

MPLS (Multiprotocol Label Switching)	Eigenständiges Netzwerk, in dem Weiterleitung anhand von Labels geschieht. Router entscheiden nicht mehr bei jedem Paket anhand der IP-Adresse, sondern werten Label aus. Beliebige Daten können in einem MPLS-Netzwerk versendet werden. Erfordert vorherige Pfadsignalisierung und Übersetzungen an den Netzwerkgrenzen. Etabliert verbindungsorientierte Übertragung, bei der Ströme jedoch aggregiert werden können.
NSIS (Next Steps in Signaling)	Ist der Versuch unterschiedliche administrative Domänen, in denen unterschiedliche QoS-Lösungen existieren, miteinander zu koppeln. Es etabliert ein generisches QoS-Signalisierungsprotokoll und Möglichkeiten NAT- und Firewallkomponenten besser zu integrieren.

Tabelle 2.4: Gängige QoS-Architekturen auf Netzwerkebene.

2.6 ZEEBUS

Im Rahmen der Untersuchung von XMPP als Infrastruktur für föderative IoT-Szenarien arbeiten die Universitäten TU Dresden, RWTH Aachen University und die BTU Cottbus-Senftenberg in einem Forschungkollektiv zusammen. Das kollektive Forschungsthema ist: In IoT-Szenarion sollte jedes Gerät in der Lage sein, andere adäquate Geräte zu finden und einen sicheren Zugang zu diesen aufzubauen, um Informationen oder Steuersignale zu übermitteln/erhalten. Jeder Partner stellt für seinen Bereich heterogene Sensorstrukturen bereit. Diese sind über einen XMPP-Serververbund gekoppelt. Damit kann jede Universität auf Sensordaten der anderen zugreifen. Dafür werden eigenständig Nutzerapplikationen entwickelt. Die Bandbreite reicht von einfachen XMPP-Clients über mobile Applikation bis hin zu webbasierten Widgets. Zur Verfügung gestellt werden primär Wetterdaten (vgl. Tabelle 2.5).

Die BTU Cottbus-Senftenberg bewertet die generelle Realisier- und Zweckmäßigkeit von XMPP für den Zugang zu Sensornetzwerken. Cottbus untersucht Möglichkeiten XMPP und XMPP-Anpassungen (wie Chatty Things*) auf einfachen, stark eingeschränkten Geräten einzusetzen. An der TU Dresden liegt der Schwerpunkt aktuell in der Entwicklung eines Sensordatenerzeugers.

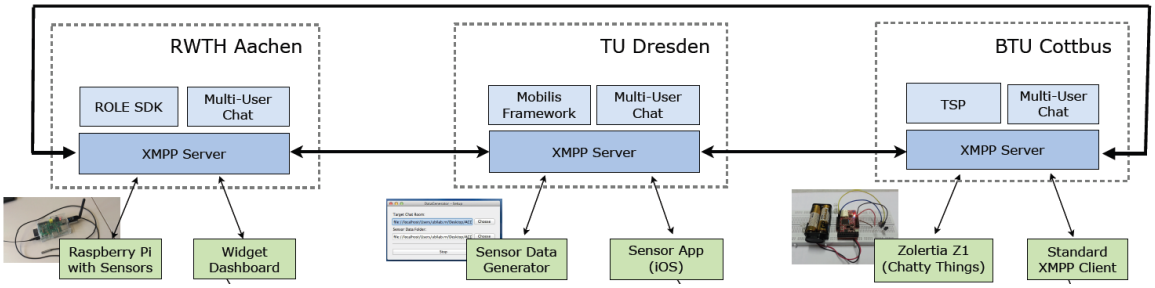


Tabelle 2.5: Kommunikations-Architektur von ACDSence. Föderation von XMPP-Servern und gemeinsamen Datenraum.

Diese Komponente verarbeitet während des Orkans *Katrina* aufgezeichnete Wetterdaten. Der Datensatz umfasst 9503 Wetterstationen und einen Zeitraum von 10 Tagen. Jede Wetterstation wird als Sender emuliert, dem eine beliebige Anzahl heterogener Empfänger zugeordnet werden kann. Die Emulationssoftware erlaubt es, den zeitlichen Verlauf zu beschleunigen. Dadurch sind Stresstest und Effizienzanalyse der Kommunikationsinfrastruktur möglich.

ACDSense ist ursprünglich mit XMPP realisiert. Ein Schwerpunkt ist es, ACDSense so zu erweitern, dass MQTT als Kommunikationsprotokoll unterstützt wird. Ferner soll

eine Evaluation zeigen, wo die Vor- und Nachteile beider Protokolle zu finden sind.

3 Verwandte Arbeiten

Wie die Grundlagen bereits zeigen, sind die Arbeitsfelder mit einem Bezug zu IoT vielzählig. Ein Schwerpunkt auf Kommunikationsprotokolle lässt eine Inselbetrachtung einzelner Komponenten kaum zu, denn ein Protokoll muss immer Ende-zu-Ende bewertet werden. Daher ist es notwendig, jede Schicht von der Anwendung bis zur physikalischen Übertragung und vom Sender über die Infrastruktur bis hin zum Empfänger zu berücksichtigen. Nicht zuletzt bestimmt das Szenario dessen Anforderungen und das Wesen der übertragengen Daten das Maß an Geeignetheit eines Protokolls mit.

Die hier vorgestellten Arbeiten sollen daher Aspekte bedienen, die in der eigentlichen Arbeit zu kurz kommen, um ein ausreichend vollständiges Wissen voraus zu setzen. Hinzu kommen Aussagen, die für die spätere Analyse als gegeben angenommen werden.

3.1 Szenarien und Anwendung

In der ersten Kategorie verwandter Arbeiten lassen sich Beispiele für typische IoT-Szenarien einordnen. Dazu zählen insbesondere jene, die einen entsprechenden Bezug zu den vorgestellten Protokollen haben. [Chri00] zeigt, wie ein PKW-Verleih mit Endnutzergeräten gekoppelt werden kann, indem Fahrzeugdaten für den Verleih, wie für den Kunden über mobile Anwendungen mit Hilfe von *OData* zur Verfügung gestellt werden. *Gorden Hunt* hat eine ganze Reihe von DDS-basierten Lösungsvorschlägen für typische Szenarien in einen Foliensatz zusammengefasst [Gord07]. In [WiFr09] wird ein Labor mit einer auf MQTT basierenden Kontroll- und Monitoringsoftware ausgestattet. Wie DDS in einem mobilen Kontext zu integrieren ist, nämlich einem landesweiten und netzübergreifenden Flottenmanagement-System, zeigen David et al. [DVAA13]. Eine XMPP-Infrastruktur zur dynamischen Bereitstellung verteilter Dienste beschreiben Bendel et al. [BSSS13]. Bazzani et al. etablieren eine XMPP-Middleware zur Lösung von E-Health-Fragen [BCSS12]. Andere Bemühungen setzen auf CoAP [KRES14, S.-]. Besonders hervorzuheben ist ein Beitrag der *Internet of Things Initiative*. Hierbei handelt es sich zwar um ein Comic-Buch, aber gerade deswegen in seiner Fülle an visualisierten IoT-Szenarien motivierend und leicht verständlich [00e].

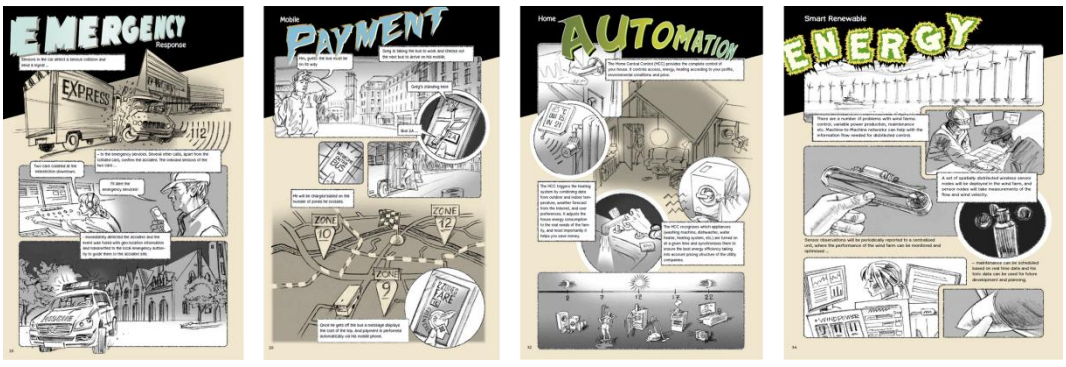


Tabelle 3.1: IoT-Szenarien als Comic illustriert [00e].

3.2 ANALYSEN

Im Analyseabschnitt konzentrieren sich die Studien auf analytische Betrachtungen von Protokollen und allgemeinen Anforderungen, die mobile und heterogene Kommunikationsumgebungen an Infrastrukturen stellen.

In [EsCG09] werden fundamentale Aussagen über Publish/Subscribe zusammengetragen. Der Fokus liegt auf Operationen, Sicherheit und Verfügbarkeit von Middlewarelösungen für zeitkritische, großskalierte Anwendungen. Daran schließ sich [HuGa04] an. Es werden mobile Verbindungen mit einbezogen, Herausforderungen erörtert und allgemein Architekturen beschrieben, um Publish/Subscribe in mobilen Strukturen zu integrieren. [PoNW07] konzentriert sich diesbezüglich auf mobile Ad Hoc Netzwerke, in denen sich Knoten/Fahrzeuge mit unterschiedlicher Geschwindigkeit bewegen. [MoLG02] erweitert die Untersuchung von mobilen Ad Hoc Netzwerken um Aspekte der Dienstqualität (QoS).

Das Forschungsfeld *Quality of Service* ist relativ dicht besetzt. Das liegt unter anderem daran, dass sich Dienstgüte-Anforderungen in jeder OSI-Schicht implementieren lassen. Einen ausführlichen Überblick über Mechanismen und Architekturen zur Sicherung von Übertragungsmerkmalen in IP-Netzen bietet [PaRR11]. Man erfährt, welche Möglichkeiten und Verbesserungen mit IPv6 hinzukommen. Die Mechanismen sind auszugweise in den Grundlagen erwähnt. Eine ausführliche Zusammenfassung bietet zugleich [00f]. [GSBV03] ergänzt gängige Methoden durch zukünftige. Nach [RoHo00] können diese Mechanismen in enger Koppellung genutzt werden, um netzübergreifende Qualitätsparameter bereitzustellen. Das Kernnetz wird von MPLS-Strömen dominiert. Die Netzränder unterstützen IntServ- und diffServ-Techniken. Eine konzeptionelle Schnittstelle beschreibt eine QoS-fähige, skalierbare Ende-zu-Ende-Übertragung. [LuSh11] gibt einen Überblick über QoS-Fähigkeiten im WLAN. ES werden aktuelle Forschungsansätze beurteilt und Herausforderungen für die Zukunft formuliert. In die Betrachtung fließen Perspektiven der Energieefizienz und Heterogenität ein. Da in Mobilfunknetzen wiederum individuelle QoS-Mechanismen installiert sind, ergeben sich dort adäquate Forschungsfelder. Ein Hauptanliegen ist auch hier eine Bereitstellung von Dienstgütemerkmalen über Netzgrenzen hinweg. [ZhWG12] erschließt die gängigen Bearer-Techniken innerhalb der Funknetze und arbeitet eine netzübergreifende Lösung auf Basis von IPv6 Flow Label aus. Einen ähnlichen Versuch wagt [BFLM11], nutzt dafür jedoch ein eigenes QoS-Signalisierungsprotokoll. Es zeigt im Vergleich bessere Eigenschaften gegenüber SIP. Die Lösung ist insbesondere für M2M-Szenarien konzipiert.

Im Kontext von Protokollen analysiert [AnMO12] das Verhalten von UDP und TCP in IEEE-802.11-Netzen. In [SGCI12] wird eine Framework vorgestellt, mit dem eine transparente Handoverunterstützung mit Hilfe von *mSCTP* realisiert wird. [Isod14] listet eine Reihe von Whitepapern auf, welche XMPP, speziell XMPP-Server, in verschiedenen Netzwerkkonfigurationen analysieren. [KGKS11] integrieren XMPP in ihre Sensor-umgebung und zeigen Auslieferungszeiten für UMTS-Netzwerke. [BaBh13] analysieren CoAP und MQTT in Netzwerken mit simuliertem Paketverlust. [LKHJ13] konzentrieren sich dabei primär auf MQTT und dessen QoS-Parameter. Im Vergleich MQTT zu CoAP bewirken geringe Verlustraten geringere und hohe Verlustraten höhere Übertragungslatenzen [14b]. DDS gilt als *low latency*-Kommunikationsarchitektur. Daneben unterstützt DDS einen umfangreichen Katalog an QoS-Fähigkeiten, wie in [Pard03] beschrieben. Auf der anderen Seite zeigen sich für DDS Schwierigkeiten in *large scale*-Systemen. Das liegt zum einen an der Kommunikation, die auf IP-Multicast aufbaut und voraussetz. Zum anderen zeigen sich Probleme bei der Neuübertragung im Zusammenhang mit vielen

Empfängern. Hinzu kommt eine mangelhafte Unterstützung heterogener Netzwerke, besonders über Netzwerkgrenzen hinweg [Espo00]. Einige Arbeiten versuchen diese Nachteile zu beseitigen, indem Funktionen ausgelagert werden. In [DVAA13, S.-] gibt es an den Grenzen zu mobilen Netzwerken sogenannte Gateways. Diese übernehmen die Kommunikation zu mobilen Knoten, indem sie für die Übertragung MR-UDP anstatt DDS nutzen. MR-UDP ist ein zuverlässiges auf UDP-basierendes Protokoll, das IP-Änderungen toleriert. Einen ähnlichen Ansatz verfolgen [CoFN10] mit REVENGE, einer auf DDS basierenden Middleware zur Kommunikation über Netzwerkgrenzen hinweg. Eine Vielzahl von Arbeiten konzipieren eigene Middlewarelösungen [ChLK08].

3.3 GEGENÜBERSTELLUNGEN

Funkverbindungen unterscheiden sich in wesentlichen Dingen von üblichen kabelgebunden Netzwerken. Diese Unterschiede wirken sich auf höhere Schichten aus. Das erfordert oftmals die Anpassung etablierter Protokolle. Für TCP gibt es eine Reihe von Erweiterungen, die den Anforderungen mobiler Übertragungen genügen sollen. Einen Überblick und Vergleich dieser findet sich in [BaAL06]. [GoKa13] nehmen sich ebenfalls TCP vor und vergleichen es mit seinem direkten Konkurrenten SCTP. Die Untersuchung wird im Verlauf auf die Techniken Mobile IP, HMIPv6, FMIPv6 und mSCTP ausgeweitet. Die Autoren kommen zum Schluss, dass mSCTP die Performance bei Handover verbessert, Datenraten erhöht und Bandbreiten optimaler nutzt.

Features	MIP	HMIPv6	FMIPv6	mSCTP
Operating layer	Network	Network	Network	Transport
IP Connections	One	One	One	More than One
Packets Reordering	Very low	Very low	Minimum	Required
Concurrent multi-path transfer	Not Possible	Not Possible	Not Possible	Possible
Type of Handover	Hard	Hard	Hard	Soft
Packet Loss	High	High	High	Least in Comparison
Handover Latency	Bad	Moderate	Very Good	Very Good
Impact of speed of MN on Throughput	Bad	Bad	Very Bad	Least in Comparison
Location Management	Provided	Provided	Provided	Not Provided

*

Im Feld QoS-fähiger Middleware erarbeiten [DSES11] eine Zusammenfassung und Bewertung von verbreiteten Implementierungen (darunter CORBA, ZeroMQ, DDS). Eine abstraktere Analyse ereignisgetriebener Middleware mit QoS-Unterstützung findet sich in [MaKB07]. Die Autoren entwerfen eine Taxonomie, die sich nach der topologischen Struktur, der Fähigkeit, Knoten im Kommunikationsnetz zu finden und dem Zugriff auf darunter liegende Schichten richtet.

In [00g] werden DDS, CORBA, JMS und Web Services gegenübergestellt und auf charakteristische Merkmale von DDS eingegangen. Der Artikel [Schn00] zählt

grundsätzliche Merkmale von MQTT, XMPP, DDS und AMQP auf. MQTT ist dort geeignet, wo eine große Zahl von Geräten Informationen in eine gemeinsame Infrastruktur speisen, zum Beispiel zur Überwachung und Kontrolle. Die Stärken von XMPP liegen in der Adressierung, Sicherheit, Skalierbarkeit und die Schwächen im Umgang mit großen Datenmengen. XMPP ist ideal für verbraucherorientierte Szenarien. DDS eignet sich für Systeme, in denen Geräte direkt mit Geräten kommunizieren. Aufgrund der Architektur sind diese Systeme notgedrungen geschlossen. AMQP wird aufgrund des verlässlichen transaktionellen Nachrichtenschemas als Protokoll für die Serverkommunikation untereinander angesehen. Analog vergleicht [Sina00] in ihrer Studie XMPP, MQTT, AMQP, OData, CoAP und PubSubHubbub.

MQTT, Sekunden*

DDS, Mikrosekunden

XMPP, Sekunden

AMQP, Sekunden

3.4 Optimierungen

Der Großteil der verwandten Arbeiten experimentiert damit, bekannten Problemen auf Netzwerkebene durch Erweiterungen oder eigene Protokollentwürfe zu begegnen. [LOGM10] optimiert Protokolle mit Hilfe eines Frameworks zur Findung optimaler ARQ-Strategien. [YoAT10] stellt einen Mechanismus vor, um Ströme kurzer Länge und geringer Bandbreite besser zu handhaben. [KJPR12] enthält eine Architekturbeschreibung zur netzwerkübergreifenden QoS-Generalisierung. Ähnliche Anstrengungen berücksichtigen insbesondere mobile Netzwerktechnologien [ShJa11]. In heterogenen komplexen Netzen muss der Zustand kontinuierlich ermittelt werden, um auf QoS-Veränderungen reagieren zu können. Diesem Thema widmen sich [ElSa12, STKA13]. Eine entsprechende Softwareplattform für Android ist in [KuAn11] beschrieben.

Auf höheren Ebenen zählt speziell SCTP zu den Protokollen, das auf sein Potential für mobile Umgebungen getestet wird. SCTP ist im Kern nicht optimal an die Anforderungen angepasst, sodass Verbesserungen notwendig sind [KeSB06][TuRi00]. Für XMPP gibt es eine Sammlung von Erweiterungen, die versuchen, explizit das *Internet der Dinge* abzudecken [00h].

3.5 Abgrenzung

Wird die Summe aller in einem Bezug zu IoT stehenden Arbeiten betrachtet, dann fällt auf, dass konkreten Analysen von Anwendungsprotokollen fehlen, es jedoch eine Vielzahl an Versuchen gibt, eigene Middlewarelösungen zu konzipieren oder bestehende Protokolle um eine individuelle Fähigkeit zu erweitern.

- Erweiterungen schränken Interoperabilität ein
- Neue Entwicklungen setzen sich kaum durch
    - IoT Vernetzung vieler Domänen, über Netzwerkgrenzen hinweg
- Erster versuch SCTP
    - Multihoming, multistreaming
    - Windows kein nativer SCTP unterstützung
    - Linux native unterstützung
    - Android per Kernel rebuilt

    - wenig potential
    - zu hoher aufwand
- zweiter versuch QoS aware UNderlay
    - underlay awarness
    - QoS fähigkeiten von android
    - Monitoring von stats
    - Viele funktoinnen nicht untertützt
        - Abhängig von hersteller
        - Abhängig vom betreiber
        - Mobilfunknetz abgeschottet
- Dritter versuch MR-UDP
    - MR-UDP/TCP Proxy
    - MR-UDP code nicht open source
    - MR-UDP selber funktioniert, eingesetzt in Framework kamen nur NULLsegment nachrichten an

4 ANALYSE

Primäres Ziel der Arbeit ist die Untersuchung und Bewertung etablierter Protokolle, die in enger Verbindung zum Internet der Dinge stehen. Dabei betrifft es vorwiegend Protokolle der Anwendungsschicht. Die wichtigsten wurden bereits im Grundlagenkapitel vorgestellt (vgl. 2.2).

Entsprechend den OSI-Schichten wird ein Protokoll einer adäquaten Ebene zugeordnet. Es kann jedoch nicht ausschließlich innerhalb dieser Schicht bewertet werden, da ein Protokoll auf darunter liegende Schichten aufbaut beziehungsweise von diesen beeinflusst wird. Im Kontext einer Protokollbewertung erweisen sich sowohl das bestehende OSI-Modell, wie auch das TCP/IP Modell als inkonsistent in ihrer Belegung. OSI bewertet TCP und UDP als Protokolle der Transportschicht. Jedoch weist UDP praktisch keine Funktionalitäten der Transportschicht auf. Hingegen zeigen IP und UDP viele Ähnlichkeiten. Der eigentliche Unterschied ist, dass UDP über Ports konkrete Dienste adressieren kann. TCP/IP kürzt alle Funktionalitäten oberhalb der Transportschicht in eine Anwendungsschicht zusammen und bewertet UDP und TCP ebenfalls als gleichwertig. Es wird daher ein zweckmäßigeres Modell vorgeschlagen.

* Protokolle eintragen

Schicht	Operationen	Protokolle
6	Operationen der Anwendung	Operationen mit bekannter Semantik
5	Operationen der Middleware	HTTP, SOAP, FTP, TLS, DTLS
4	Zuverlässigkeit, Flusskontrolle, Paketzusammenführung	SCTP, TCP, TRP, DTLS, PGM
3	Paketauslieferung, Routing	UDP, IP, IP-Multicast

Tabelle 4.1: Geeigneteres Schichten-Modell zur Klassifizierung von Protokollen

Im Modell (Tabelle 4.1) befinden sich Operationen, welche unmittelbar mit dem Benutzen der Middleware verbunden sind, in der obersten 6. Schicht. Dazu zählt zum Beispiel der Aufruf zum Versenden einer Nachricht. Die Schicht darunter bündelt das Kommunikationsprotokoll beziehungsweise die Middleware in sich. Fast alle hier betrachteten Protokolle lassen sich in diese Schicht einordnen. Ein Kommunikationsprotokoll operiert immer auf einem Protokoll der 4. Schicht. Adäquat gilt für Protokolle der 4. Schicht, dass diese immer auf mindestens ein Protokoll der darunterliegenden 3. Schicht aufbauen. DTLS ist ein Beispiel für ein Multischichtenprotokoll und lässt sich in die Schichten drei und vier einordnen. In der Tabelle sind alle Protokolle gelistet, welche während der Bearbeitungszeit betrachtet wurden. ~~Einen vollständigen Überblick gibt Tabelle * im Anhang~~.. Da der Aufwand einer umfangreichen Analyse aller Protokolle den Rahmen der Arbeit sprengt, ist eine geeignete Abgrenzung notwendig.

4.1 ABGRENZUNG

Im Kontext von IoT interessieren insbesondere jene Protokolle, welche sich unmittelbar zur Realisierung der betrachteten Szenarieneignen. Das sind Protokolle der 5. Schicht, da jene ausreichend Funktionalitäten bereitstellen. Die 4. Schicht wird nicht näher betrachtet. Oftmals ist diese tief im Betriebssystem verankert und ferner gibt es eine viel höhere Zahl verwandter Arbeiten, die sich diesem Thema widmen *.

Des Weiteren werden Web-orientierte Protokolle ausgeklammert. Diese sind nach der vorherrschenden Sicht auf das Internet, speziell aus der Sicht des Browsers, entworfen. Nicht selten ist das Protokoll eng an einen Dienst gekoppelt und nicht selten wird zudem ein Webserver benötigt, wodurch diese Techniken nicht sonderlich leichtgewichtig sind.

Zu guter Letzt trennt sich die Analyse von STOMP und ZeroMQ. STOMP ist zwar ein einfaches Protokoll, aber ebenso einfach bezüglich fehlender Funktionen. Alle Daten werden als Text behandelt. Das führt zu Schwierigkeiten bei der Serialisierung und ist nicht sonderlich effizient. Ferner ist STOMP nachrichtenzentriert. Es fehlen oftmals Zustandsinformationen über die Verbindung selbst. ZeroMQ ist eine flexible, performante, asynchrone Messaging-API und ist deswegen geeignet, eine Kommunikationsmiddelware genau nach Anforderung zu entwickeln. Dafür stehen eine große Vielfalt und Protokollunterbauten bereit. Für den Einsatz müssen jedoch zusätzlich Operationen der Anwendungsebene eigenständig implementiert werden. Dennoch lohnt es sich in späteren Arbeiten das Potential von ZeroMQ zu erfassen.

4.2 Protokollauswahl

Für die Analyse verbleiben somit MQTT, XMPP, CoAP, DDS und AMQP. Studiert man aktuelle Literatur oder sucht im Allgemeinen nach dem, was als *State of the Art* gilt im Kontext von IoT und Anwendungsprotokollen, wird man auf die genannten stoßen [Tark12, Puza00, Mhol00].

Mit Hilfe der in den Grundlagen beschriebenen Kriterien werden Eigenschaften für jedes Protokoll zusammengefasst (siehe Tabelle 4.2). Es ist nicht notwendig auf jedes Detail einzeln einzugehen. Es werden daher nur entscheidende und einflussreiche Charakteristiken herausgestellt.

XMPP ist ursprünglich *als Instant Messaging* Protokoll konzipiert. Spezifikation und Umfang sind daher an dafür typische Funktionen angepasst. XMPP unterstützt beispielsweise Abwesenheitsnachrichten. Um das Protokoll für viele Nutzerszenarien einsetzen zu können, ist ein Kernaspekt von XMPP die integrierte Erweiterbarkeit. Dafür setzt man auf XML als Markup-Sprache und demensprechende *Interpreter*. Daher existiert über der eigentlichen Übertragungsschicht eine Interpretationsschicht. Zwar erweist sich das Protokoll dadurch als äußerst flexibel, auf der anderen Seite wird es komplex, besonders in Hinsicht auf Serialisierung und Fehlertoleranz.

Zwei der fünf Protokolle, nämlich MQTT und AMQP, spezifizieren keine Discoverymechanismen. Das bedeutet, dass es keine Möglichkeiten gibt, Teilnehmer oder Anwendungsressourcen dynamisch im Kommunikationsnetz zu ermitteln. Diese müssen bereits vor der Kommunikation bekannt sein. Alternativ können Brokerimplementationen unabhängig vom Protokoll Discoverytechniken bereitstellen. DDS und CoAP sehen in ihrer Spezifikation vor, Teilnehmer im Netzwerk über IP-Multicast zu erfassen. In DDS wird dieser Ansatz zusammen mit RTPS kombiniert, um DDS spezifische Ressourcen zu finden [10b, S.-]. CoAP setzt auf eine Erweiterung (*CoRE Link Format),* um Ressourcen von einem CoAP-Knoten abzufragen [Zach00]. Da XMPP eine zentrierte Architektur etabliert, erfolgt die Discoverybehandlung auf dem Server selbst. *XEP-0030: Service Discovery* spezifiziert die dafür notwendige Erweiterung [HMES08].

Wie bereits erwähnt, kommuniziert XMPP über eine zentrale Instanz. Das gleiche gilt für MQTT und AMQP, wobei XMPP und AMQP eine Multibroker-Architektur nutzen und damit Netze (Server-Server) von Sternen (Client-Server) erzeugen.

MQTT hingegen implementiert eine Singlebroker-Architektur und damit einfache Sterntopologien. CoAP und DDS setzen auf eine dezentrale Kommunikation. DDS erzeugt* ein reines P2P-Netz. Darin hält jeder Knoten selbst die zu teilende Information und das Wissen über das Netzwerk mit den Adressen und Informationen anderer Knoten. CoAP verlangt keine strikte Architektur oder Topologien. P2P-Netze sind gleichermaßen möglich, wie zentralisierte Server/Proxy-Server oder Gateways zu anderen Netzen.

CoAP und DDS setzen auf UDP. Dadurch sind nicht nur logische Multicastnachrichten möglich, sondern diese können dank IP-Multicast effizient auf Netzwerkebene umgesetzt werden. Dadurch wird vermieden, eine Nachricht vom Sender aus sequentiell an alle Empfänger zu verschicken. Von IP-Multicast-fähigen Knoten können simultan Pakete an verschiedene Empfänger dirigiert werden. Beide Protokolle nutzen IP-Multicast zur Discovery von Kommunikationsknoten. Es gibt allerdings Einschränkungen von IP-Multicast über Netzwerkgrenzen hinaus. Hinzu kommt, dass in einigen mobilen Strukturen, wie zum Beispiel in Mobilfunknetzen, Multicastverbindungen unterbunden werden.

	XMPP	MQTT	CoAP	AMQP	DDS
Beschreibung	XML-Routing Protokoll, integrierte Erweiterbarkeit	MOM für Topic-basiertes Pub/Sub	pull-basierte REST Schnittstelle, Pub/Sub über Token	QOM für Queue-basiertes Pub/Sub	Echtzeitkommunikations - Architektur
Anwendung	Instant Messaging	M2M, Home Automation, Facebook Messenger	Eingeschränkte Netzwerke, M2M, Home Automation	Server2Server Kommunikation, Cloud-Dienste	Echtzeit-kommuni-kation in verteilten Systemen, industrielle Automation
Kommunika-tion	Knoten-orientiert; Zentralisiert mit Pub/Sub-Erweiterung	Topic-orientiert; Zentralisiert, Pub/Sub	Ressourcen-orientiert; dezentrales Req/Res; optional Pub/Sub	Queue-orientiert; Zentralisiert, Pull oder Pub/Sub	Daten-orientiert; dezentrales Pub/Sub
Topologie	Stern-Stern; Client-Server, Server-Server	Stern; Client-Server	Hybrid; P2P	Stern-Stern; Client-Server, Server-Server	Netz; dezentrales P2P
Routing	Unicast, Multicast	Unicast, Multicast	Unicast, Multicast, Broadcast	Protokollrouting über Exchanges; Unicast, Groupcast, Multicast	Unicast, Multicast, Broadcast
Discovery	Erweiterung, generisches Discovery für XMPP-Entitäten und Items	-	Knoten durch UDP Multicast, Ressourcen durch Directory-Prinzip	-	Knoten und Ressourcen über UDP-Broadcast
Transport	TCP, HTTP, Jingle (beliebiges Protokoll)	TCP	UDP, DTLS , 6LoWPAN	TCP	RTPS über UDP
Standard	offen, IETF	offen, OASIS	offen, IETF	offen, OASIS	offen, OMG
Adressierung	Knoten-JID	Topics und Client-ID	URI und IP	IP vs. Queue/Exchange?	Topics und IP

Tabelle 4.2: Zusammenfassung der Anwendungsprotokolle

~~Protokolle können nicht isoliert verglichen werden. Sie müssen beurteilt werden im Kontext vom Dienstmodell (Service Model). Dienstmodell und Protokoll sind miteinander gekoppelt. Hinter dieser etwas umständlichen Beschreibung verbirgt sich die Idee der Middleware. Das Protokoll selbst befähigt noch nicht zur Kommunikation. Erst die Implementation stellt das eigentliche Werkzeug bereit. So ist das verwendete Protokoll selbst in der Regel eine Middleware. Deren Dienstmodel besteht aus Kommunikationsmodell, Objektmodell und Architekturmodell, welche miteinander interagieren.~~

~~Das Kommunikationsmodell ist die Abstraktion der Art und Weise, wie Anwendungen interagieren. Dies kann geschehen über Queues, Datenverteilung, Datenreplikation, Verteilten Transaktionen, *Remote Methode Invocation*. Das Objektmodell sind Middlewarekomponenten, über die Anwendungen mit dem Dienst kommunizieren, beispielweise Queues, Publisher, Domänen, Caches, Föderationen, Remote Objects und viele weitere. Das Architekturmodell bestimmt den Grad der Zentralisierungen, ob es sich um eine Brokerbasiertes Konzept oder eine Peer to Peer-Umgebung handelt. In zentralisierten Umgebungen wird das Dienstmodel im den zentrale Broker gebündelt.~~

Für die weitere Analyse betrachten wir die Protokolle in ihrem Anwendungskontext, demnach dort, wo sie eingesetzt werden sollen. Die Szenarien aus dem Einleitungsteil bilden hierfür die Grundlage und werden nach deren konkreten Anforderungen untersucht (vgl. 1.2). Wir erinnern uns an die fünf Fragen:

- Wer überträgt?
- Wo/Wohin wird übertragen?
- Wie wird übertragen?
- Wann wird übertragen?
- Was wird übertragen?

Die Antworten für jedes Szenario sind in Tabelle 4.3 zusammengefasst. Es folgt eine Anforderungsanalyse.

	Parkplatzsuche	**Notfalladministration**	**Elektromobilität**
Wer	Sensor am Parkplatz, Fahrer, Anfrage- und Auswertungsdienst	Sensoren im Krankenwagen, Notfallleitstelle	Fahrzeugsensoren, Ladestationen, Dienst
Wo/Wohin	Parkplatz an Dienst, Dienst an Fahrer, Fahrer an Dienst	Fahrzeug an Leitstelle, Leitstelle an Fahrzeug, Leistelle an weitere Stationen	Fahrzeug an Dienst, Dienst an Fahrzeuge, Stationen an Dienst, Dienst an Stationen
Wie		*Aufgabe des Protokolls*	
Wann	Bei Statusänderung eines Parkplatzes, bei Suchanfragen des Fahrers, periodisch die Position des Fahrers	Kontinuierliche Übertragung von vitalen Sensordaten, periodische Übermittlung der Position, unregelmäßige Anweisungen	Periodische Positionsübermittlung, bei Statusänderungen der Ladesäulen
Was	Position, Parkplatzstatus, Gesuche	Patientendaten, Position, Anweisungen	Positionen, Routen, Zustände, Verträge

Tabelle 4.3: Zusammenfassung von IoT-Szenarien (vgl. 1.2).

4.3 Umgebung

Über die Szenarien hinweg wird die Heterogenität der beteiligten Entitäten deutlich. Unterschiedliche Geräte kommunizieren miteinander. Die Bandbreite reicht von kleinen einfachen Sensoren mit eingeschränkten Ressourcen bis hin zu komplexen Kontrollzentren. Erstere dienen hauptsächlich dazu, Informationen wie Belegungszustände oder Messdaten zu übertragen. Letztere müssen mit einer Vielzahl an Informationsquellen und Datenstrukturen umgehen können. Zum

anderen nehmen die Entitäten unterschiedliche Rollen ein. Sender publizieren ihre Information nach außen. Das kann aktiv geschehen oder durch Anfragen initiiert werden. Empfänger interessieren sich für diese Informationen. Diese können ebenso aktiv Anfragen absenden oder asynchron informiert werden. In einigen Systemen können Kommunikationsteilnehmer ihre Rolle wechseln beziehungsweise beide Rollen einnehmen. Auf dem Weg von Sender zu Empfänger nimmt die Information entweder einen direkten Weg über das Netzwerk oder wird durch eine Vermittlerinstanz beziehungsweise einen Dienst verarbeitet.

Die Heterogenität setzt sich in dem Bereich der zu unterstützenden Plattformen und Schnittstellen fort (software- wie hardwareseitig). Je nach Ausstattung und Anforderung sind verschiedene Betriebssysteme und Standards vorzufinden. Anwendungen können auf Smartphones, eingebetteten Systemen oder hoch belastbaren Serverstationen laufen. Dort befinden sich zuverlässige Hochgeschwindigkeitsnetze, wohingegen mobile Geräte mit den Einschränkungen von Funkverbindungen zu kämpfen haben.

Diese zeichnen sich durch schwankende Übertragungsparameter, Interferenzen sowie Verbindungsunterbrechungen und der zeitweisen Nichterreichbarkeit von Kommunikationsteilnehmern aus. Besonders kritische Ereignisse sind Handover, bei denen ein Gerät zwischen Zugangspunkten zum Netz wechselt.

Die Szenarien zeigen eine hohe Dynamik in der Kommunikationsstruktur. Es gibt selten klassische Eins-zu-Eins-Beziehungen. Die Relationen zwischen Sendern und Empfängern beziehungsweise Teilnehmern und Ressourcen sind variabel. Es können Teilnehmer hinzukommen und abspringen. Hochfrequente Änderungen in der Topologie sind zu erwarten. Es bedarf Mechanismen zur autonomen Verwaltung und Erfassung von Entitäten.

Nicht zuletzt gibt es unterschiedliche Anforderungen an die Kommunikation selbst. Diese Anforderungen können sich stochastisch ändern. In dem einen Fall geht es darum, große Datenmengen, in dem anderen Fall, Nachrichten in kurzen Abständen und mit hoher Geschwindigkeit zu übertragen. Kritische Informationen müssen sicher und zuverlässig beim Empfänger eintreffen. Die Kommunikation muss daher variable Dienstgüteanforderungen unterstützen.

Die gelisteten Merkmale werden maßgeblich vom Kommunikationsprotokoll bestimmt. Das Protokoll gibt den Rahmen vor, *wie* Daten übertragen werden. Daher kann die Frage (vgl. Tabelle 4.3) nur in Kombination mit dem Protokoll selbst beantwortet werden. Um diese Fragen beantworten zu können, werfen wir einen gezielten Blick auf die Kommunikation.

4.4 Kommunikation

Insbesondere im Notfallszenario ist es von Bedeutung, dass Daten zuverlässig übertragen werden. Eine Auslieferungsgarantie und die Integrität der Daten müssen sichergestellt sein. Hinzu kommt die Notwendigkeit, dass die Auskunft über den vitalen Zustand des Patienten mit so geringer Verzögerung wie möglich übermittelt wird. Viele der Sensoreinheiten sind mit begrenzten Ressourcen ausgestattet und erfordern eine leichtgewichtige Kommunikation. Bei einer stadtweiten Suche nach Parkplätzen aktualisieren eine große Zahl von Sensoren unregelmäßig deren Zustandsinformationen. Des Weiteren treffen kaum vorhersehbar viele Suchanfragen auf Dienstebene ein. Die Relationstopologie ist extrem dynamisch. Das erfordert ein hoch skalierbares System. In einem komplexen heterogenen System können sich zu jeder Zeit Technik und Plattform

der Teilnehmer ändern. Dies erfordert eine lose Kopplung auf der technischen Seite der Kommunikation. Nicht zuletzt sind Komplikationen in inhomogenen Netzen schwer vorhersehbar. Es bedarf einer Reihe von Mechanismen, um effektiv auf Fehler zu reagieren. Eine viel diskutierte Frage ist die Sicherheit der Kommunikation von vertraulichen Daten. Tabelle 4.4 listet die wesentlichen Anforderungen und Ausprägungen auf.

Anforderung	Ausprägung
Zuverlässig	Garantierte Auslieferung Garantierte Reihenfolge Datenintegrität
Leichtgewichtig	Geringer Overhead Effiziente Serialisierung Geringer Ressourcenbedarf
Skalierbar	Teilnehmer Nachrichten Datenmengen
Echtzeitfähig	Geringe Latenzen Geringe Antwortzeiten
Lose gekoppelt	Portierbar Flexibel Plattformunabhängig
Fehlertolerant	Fehlererkennung Fehlerbehandlung Verfügbarkeit
Sicher	Vertraulichkeit

Tabelle 4.4: Anforderung an Kommunikation.

Im nächsten Schritt ist zu untersuchen, wie diese Anforderungen konkret von den einzelnen Protokollen berücksichtigt werden.

Mobile Verbindungen gelten allgemein als unzuverlässiger als drahtgebundene Netze. In Mobilfunk, WLAN oder drahtlose Sensornetze, Paketverluste können überall auftreten, sei es durch Überlagerung von Frequenzen, Überlastung oder Verbindungsabbrüchen. Um Zuverlässigkeit auf Anwendungsebene zu erreichen, benötigt man Zustandsinformationen über gesendete Nachrichten. Damit lässt sich die Auslieferung sicher stellen. Alle fünf Protokolle spezifizieren Möglichkeiten für eine zuverlässige Übertragung. Unterstützt werden *at least once*- und *exactly once*-Semantiken. Mit Ausnahme von XMPP verankern alle Protokolle diese Semantiken bereits im Kern. Für XMPP steht eine Erweiterung zur Verfügung [KSHF11].

Im Internet der Dinge stehen viele Entitäten in einer Relation und tauschen Informationen miteinander aus. Die Systeme müssen dementsprechend skalierbar sein. MQTT, XMPP und AMQP sind zentralisierte Broker-gestützte Strukturen. MQTT ist eine *Single Broker*-Architektur. Mit steigender Kommunikationslast steigt der Ressourcenbedarf bis zu einer physikalischen Grenze, ab der der Broker überlastet ist. Das Protokoll spezifiziert keine Skalierung der Brokerstruktur. XMPP und AMQP sind dagegen in der Lage, Brokerinstanzen zu *clustern* und damit horizontal zu skalieren. Bei dezentralen P2P-Netzen wie CoAP und DDS erhöht sich der Aufwand auf der Clientseite, da dort Netzinformationen gespeichert werden. Entsprechend erhöht sich bei einer Skalierung und Synchronisierung das Nachrichtenaufkommen im Netz und dezentrale Netze in Kombination mit IP-Multicast neigen zu unvorhersehbaren Verkehrsschwankungen. Reine P2P-Netzte existieren oft in lokalen Netzen, da Verbindungen über Netzwerkgrenzen hinweg kompliziert zu handhaben sind.

Ein Protokoll erzeugt Headerinformationen bei der Datenübertragung für die Steuerung von Datenströmen. Diese sollten im Verhältnis zu den transportierten Daten so kompakt wie möglich sein. Ferner sollte der Ressourcenverbrauch beim Parsen und bei der Verarbeitung niedrig bleiben. Headerinformationen von MQTT, AMQP, CoAP und DDS werden am Anfang einer Nachricht durch bits und bytes belegt. Je nach Protokoll und unterstützten Funktionen kann der Header unterschiedlich groß ausfallen. Für einige MQTT-Nachrichten ist der Header lediglich 2 Byte lang. Dagegen findet bei XMPP jede Kommunikation innerhalb eines XML- Streams statt, der notwendige Transportinformationen enthält. Dieser Stream muss aufwendig geparst werden. Nicht zuletzt ergibt sich im Vergleich zu den anderen binären Protokollen ein signifikanter Overhead.

Des Weiteren kann sich bei XMPP die eigentliche Nutzlast erhöhen, da binäre Daten vor der Übertragung erst Base-64 kodiert und auf Empfängerseite dementsprechend dekodiert werden müssen, wodurch eine weitere Serialisierung notwendig wird. Alle anderen Protokolle übertragen Daten binär. Nach dem Header folgen die Anwendungsdaten. AMQP und DDS bieten zusätzlich an, Daten und Struktur über Typdefinitionen zu trennen.

In einigen Szenarien kann es wichtig sein, den Verlauf einer Information zu speichern, anstatt den Wert bei einer Aktualisierung zu überschreiben. Für den Verlauf muss bestimmt werden, wie lang oder wie viele Aktualisierungen ein Knoten garantiert im System halten soll. XMPP kann wie auch DDS den Verlauf beliebig bestimmen. Mit MQTT ist es immer nur möglich, den letzten aktuellen Wert dauerhaft zur Verfügung zu stellen. Für CoAP muss dies auf Anwendungsebene geregelt werden und in AMQP wird dies durch die Größe einer Queue festgelegt.

Die Priorität ermöglicht es, Nachrichten bei der Kommunikation nach ihrer Wichtigkeit zu klassifizieren, um diese bevorzugt zu behandeln oder in Konfliktsituationen, wie einer Überlastung, mit dem Verwerfen von unwichtigen Nachrichten reagieren zu können. Aktuell unterstützen nur AMQP und DDS die Vergabe von Prioritäten. Die Spezifikationen geben jedoch nicht vor, wie Daten mit unterschiedlichen Prioritäten zu behandeln sind. Dies ist Gegenstand der jeweiligen Implementation. AMQP unterscheidet neben Nachrichten ebenso Empfänger unterschiedlicher Priorität.

Fehlertoleranz ist die Eigenschaft eines Systems, seine Funktionsweise im Falle von unvorhergesehenen Eingaben oder Fehler in der Hard- oder Software aufrechtzuerhalten. In komplexen Kommunikationssystemen mit dynamischer Knotenanzahl, unterschiedlichen Schnittstellen sowie variablen Anforderungen ist die Wahrscheinlichkeit für solche Ereignisse hoch. Der Fokus liegt in dieser Arbeit auf mobilen Verbindungen. Typische Konflikte sind der Verlust von Paketen oder der ganzen Verbindung. Hinzu kommt die Möglichkeit, dass sich die Netzwerkadresse oder die Schnittstelle des Gerätes ändern kann. Welche Möglichkeiten bieten die Protokolle, adäquat mit solchen Ereignissen umzugehen? Dazu zählt zum Beispiel das Erkennen von Verbindungsabbrüchen. Für alle Protokolle gilt, dass sie Verbindungsabbrüche implizit erkennen. In der Regel geschieht dies über das Festlegen einer Zeitspanne, in der entweder eine Nachricht oder die Bestätigung einer Nachricht erwartet wird. Eine Erkennung von Verbindungsabbrüchen auf Netzwerkebene unterstützt keines der Protokolle.

Neben der Fehlererkennung unterscheidet man Mechanismen zur Fehlerbehandlung. Die einfachste Behandlung ist der Versuch, die Verbindung erneut aufzubauen. Während der Phase der Verbindungsunterbrechung müssen alle Nachrichten gespeichert werden, vorerst flüchtig und, falls die Verbindung nicht kurzfristig wieder aufgebaut werden kann, persistent. Dies ist auf Sender-, wie auch Serverseite notwendig. Eine persistente Speicherung erhält Nachrichten im Falle eines Systemabsturzes. Um auf IP-Änderungen reagieren zu können, ist es

sinnvoll, die logische Adressierung von der physikalischen Adressierung zu trennen. Eine logische Adresse identifiziert jeden Knoten eindeutig im System. Nach dem Bezug einer neuen IP bleibt die Nachrichten-Empfänger-Relation erhalten. Benötigt werden darüber hinaus Mechanismen, welche die physikalische und logische Adressierung synchronisieren.

Echtzeit	XMPP	MQTT	CoAP	AMQP	DDS
Zuverlässig-keit	Standardmäßig TCP; Erweiterungen: XEP-0184, XEP-0198	TCP; Nachrichten: most once, least once, exactly once	zuverlässiges UDP; Multicast unzuverlässig	TCP; Nachrichten: most once, least once, exactly once	Umfassende Konfigurations-möglichkeiten
Skalierung	Hybrides P2P, Clustering, abhängig von Konfiguration und Implementation	zentralisierte Struktur, abhängig von Implementation, Bridging	Dezentral, UDP, REST, IP-Multicast, Proxy, NAT-Probleme	Hyprides P2P, Clustering, lose Kopplung	dezentral, P2P, IP-Multicast, NAT-Probleme
Fehler-toleranz	+ Caching, JID, Verlauf, Presence, Reconnection - zentraler Server, TCP	+ Caching, Client ID, Zuverlässigkeit, Last Will, Persistierung - Discovery, zentraler Server	+ Proxy, Caching, Zuverlässigkeit, Discovery - IP,	+ Caching, Exchanges, Zuverlässigkeit, redundante Queues, Proxy, Reconnection - zentraler Server, Discovery, IP	+ umfangreicher QoS Katalog, dezentrale Discovery, Persistierung, Redundanz, Caching - IP, Knoten ermitteln Netzwerk-informationen
Overhead	hoch, da XML-Streaming, aufwendige Serialisierung	kompaktes Headerformat	kompaktes Headerformat	umfangreicher Header	umfangreicher Header
Payload	wird in XML-Nachricht verpackt, binäre Daten werden Base64 codiert, abhängig von Implementation	binärer Payload, max. 256 MByte pro Nachricht	binär, URI und Content-Type, Nachricht muss in IP Datagram passen, größere Nachrichten durch Erweiterungen	binär, AMQP Type Definition, beliebig groß, universales Headerformat für Routing und Flusssteuerung	binär, DDS Type Definition, beliebig groß
Echtzeit-fähigkeit	Sekunden	Millisekunden-Sekunden	Millisekunden-Sekunden	Sekunden	Millisekunden
Verlauf	beliebig konfigurierbar	letzte Nachricht	-	Queue-Größe	beliebig konfigurierbar
Priorität	-	Nachricht, abhängig von Implementation	-	Nachricht und Empfänger, abhängig von Implementation	Nachricht, abhängig von Implementation

Tabelle 4.5: QoS-Unterstützung der Protokolle

- Abgrenzung AMQP, DDS
  - DDS
    - in vielen Studien, vergleichen untersucht
    - Nur für lokale Netzt gut geeignet
    - Multicast in mobilen Netzten kaum unterstützt
    - Alle Knoten halten Informationen
  - AMQP
    - Eher ein Server-to-Server Protokoll
- MQTT, XMPP, CoAP relativ selten beleuchtet und nur oberflächig

4.5 Periodische Daten

Der letzte Teil der Analyse widmet sich den Charakteristiken periodischer Daten. Einige wurden im vorhegenden Teil bereits angesprochen. Im Kontext eines Kommunikationsprotokolls definiert die Periode die zeitliche Steuerung der Kommunikation und nicht das zeitliche Erfassen von Daten. Beides steht jedoch in Relation zueinander.

Die Periode als Parameter ist kein Merkmal der Transportschicht, da sie sich nicht direkt auf die Flusssteuerung auswirkt. Sie ist entweder in der Anwendungsschicht implementiert oder wird durch die Kommunikationsmiddleware unterstützt.

Zuallererst bedeutet periodische Kommunikation, dass es wiederkehrende Sende- und/oder Empfangsereignisse gibt. Es ist zwischen einem periodischen Bezug und einem periodischen Versenden zu unterscheiden. Beide Ereignisse treten in der Regel in zeitlich konstanten Abständen auf, welche sich in Abhängigkeit vom Anwendungskontext dynamisch verändern können. Zum Beispiel braucht ein Fahrzeug seine Position nur selten zu übermitteln, wenn es sich nicht bewegt. Den Szenarien nach sind die Mehrzahl der übertragenen Daten primitive Informationen oder Nachrichten über Zustände. Das zu übertragende Datenvolumen ist gering. Nicht selten gibt es eine lange Phase der Ruhe, in der keine Kommunikation und nur sehr kurze Phasen der Übertragung stattfinden.

Anders gesagt, kommen lange Ruhephasen vor, in denen eine aktive Verbindung hinfällig ist. Da durch das Wissen über die Periode zukünftige Sende-/Empfangsereignisse bekannt sind, können Verbindungsressourcen reserviert werden. Mit dem zusätzlichen Wissen über der Datenstruktur und die davon abgeleitete Datenmenge kann ferner der Bedarf an Ressourcen, zum Beispiel Bandbreite, abgeschätzt werden. Das Ausbleiben von Nachrichten innerhalb eines Zeitfensters plus einer tolerierten Abweichung impliziert darüber hinaus Verbindungsprobleme. Eine strengere Auslegung ist die Angabe einer *Deadline*. Diese gibt den Zeitraum an, in dem das Eintreffen einer Nachricht erwartet wird, unabhängig von der Periode.

Die Langlebigkeit oder *Time to Live* ist die Zeitspanne, in der eine Information in einem System gehalten wird. Sie beschreibt den Umstand, ob eine Information noch gültig/aktuell ist und legt ebenso fest, wie lange eine Information im System garantiert verfügbar sein muss.

Periodische Daten sind nicht zwangsläufig semantische Ereignisse, wie zum Beispiel ein Zustandswechsel. Aus der Interpretation von Datenfolgen können semantische Ereignisse generiert werden. Ein Abonnent von periodischen Daten interessiert sich demnach für

- den aktuellen Wert,
- eine Folge von Werten,
- die Interpretation von Werten (Ereignisse).

Die Anforderungen periodischer Kommunikation sind dementsprechend die Zwischenspeicherung und Bereitstellung von Werte- oder Ereignisreihen. Da nur eine endliche Folge gespeichert werden kann, muss eine Strategie im Falle eines Bufferüberlaufs definiert sein. Mögliche Strategien sind das Verwerfen des jeweils ältesten Wertes oder das Verwerfen von neuen Werten nach dem *FIFO*-Prinzip. Daran gebunden sind das Festlegen der Buffergröße oder der Queuelänge. Das Ganze kann durch das Festlegen einer maximalen Speicherdauer oder Wertegültigkeit erweitert werden. Nach Ablauf werden entsprechende Werte aus dem Buffer entfernt. Durch diese Maßnahmen können Datenerfassung und –übertragung zeitlich voneinander getrennt werden. Somit existieren eine Erfassungs- und Sendeperiode.

Dies führt zu einer anwendungs- oder middlewaregesteuerten Bündelung. Der Unterschied zum erwähnten Zwischenspeicher ist die zusätzlich gebündelte Übertragung von Nachrichten. In der Regel wird dadurch versucht, das Nachrichtenvorkommen im System zu verringern und die vorhandenen Ressourcen (Bandbreite, Energiequelle) so effektiv wie möglich zu nutzen. Gleichermaßen sind für eine autonome Bündelung (durch die Middleware) die Buffergröße und eine Auswahl- wie Verwerfungsstrategie festzulegen. Die Auswahl bestimmt nach zeitlichen, volumetrischen (Datenmenge) oder kausalen Kriterien, ob Nachrichten gebündelt oder wann diese verschickt werden. Die Verwerfungsstrategie separiert Nachrichten von einer Bündelung. Der Nachteil einer autonomen Bündelung ist, dass es zur Stauung und unvorhersehbaren Ressourcenauslastung auf Senderseite kommen kann. Je nach Anwendungsfall sollte entschieden werden, ob eine erfassungsabhängige Übertragung die bessere Wahl ist, indem zum Beispiel aus einer Menge von zwischen zwei Sendevorgängen gesammelten Nachrichten, allein die aktuelle versendet wird.

In einem Kommunikationsnetz kann es verschiedene Bereiche für die Speicherung von Daten geben. Allgemein wird zwischen der Speicherung beim Endknoten oder Vermittler und einer flüchtigen (im Arbeitsspeicher) oder persistenten (Festplatte, Datenbank) unterschieden.

Anforderung	Ausprägung
Periode	Zeitliche Automatismen, Senden durch Periode, nicht durch Aufruf gesteuert, Behandlung von Abweichungen
Bündelung	Aggregation von Nachrichten, zeitlich, volumetrisch[1], kausal[2], hybrid
Caching/Speicherung	Lokal, entfernt, temporär, persistent
Deadline	Gültigkeit der Information, Gültigkeit einer Verbindung
Time to Live/Langlebigkeit	Zeitliche Validität, Erhaltung im System

- Tabelle 4.6: Anforderung in periodischen mobilen Szenarien.

Interpretation und Mechanismen zur Bündelung von Paketen werden nur in DDS spezifiziert. Für die restlichen Protokolle ist es abhängig von der Implementation. In AMQP-Queues findet eine implizite Bündelung statt, da dort Nachrichten bis zur Empfängerabfrage gespeichert werden.

[1] Nachricht im bestimmten Größenbereich
[2] Abhängig von bestimmter Bedingung

MQTT, CoAP und DDS unterstützen Deadlines. In MQTT wird dafür ein *keep alive*-Wert in Sekunden angeben. Dadurch wird dem Server mitgeteilt, wann eine nächste Nachricht eintrifft. In CoAP gibt es das Attribut *max-age*, welches eine Zeitspanne definiert, in der ein Wert gültig ist. Daran können Empfänger erkennen, wann sie eine neue Anfrage stellen sollten.

Ähnlich verhält es sich mit der Festlegung von Perioden. DDS unterstützt Perioden sowohl auf Sender- wie auf Empfängerseite. In XMPP und AMQP gibt es dahingehend keine Möglichkeiten. Für CoAP und Mqtt kann man indirekt über die Festlegung einer Deadline ein periodisches Verhalten simulieren.

Echtzeit	XMPP	MQTT	CoAP	AMQP	DDS
Periode	-	-	-	-	Ja
Bündelung	-	-	über Proxy	in Queue	ja
Speicherung	lokal und auf Server; Verlauf	auf Client und Server; optional persistent	lokal, entfernt auf Proxy	entfernt in Queues, flüchtig, persistent	lokal, entfernt; flüchtig, persistent
Deadline	-	keep alive	max-age	?	ja
Langlebigkeit	?	retained	max-age	?	ja

Für die weitere Analyse ziehen wir Beispieldaten aus ACDSense heran. Wetterstationen veröffentlichen ihre Werte als JSON-formatierte[3] Zeichenketten. Listing 4.1 zeigt drei beliebig gewählte Datensätze.

```
{"sensorevent":{"type":"Pressure:inches","values":[27.7],
"timestamp":"2005-08-26T00:05:00-07:00"}}

{"sensorevent":{"type":"Pressure:inches","values":[27.71],
"timestamp":"2005-08-26T01:10:00-07:00"}}

{"sensorevent":{"type":"RelativeHumidity:percent",
"values":[49.0],"timestamp":"2005-08-26T01:35:00-07:00"}}
```

Listing 4.1: Typische JSON-formatierte Datensätze in ACDSense.

Die Beispiele bestätigen die Annahme von der einheitlichen Struktur periodischer Daten. In Listing 4.2 sind die Struktur blau und fett, die eigentlichen Daten rot und normal hinterlegt.

```
{"sensorevent":{"type":"RelativeHumidity:percent",
"values":[49.0],"timestamp":"2005-08-26T01:35:00-07:00"}}
```

Listing 4.2: Datenstuktur (blau/fett) und Datenwerte (rot/normal) in ACDSence.

Die Struktur ist in jeder Nachricht einheitlich und wird redundant übertragen. Sie kann daher von der Nachricht getrennt werden.

```
{"sensorevent":{"type","values","timestamp"}}

"RelativeHumidity:percent",[49.0],"2005-08-26T01:35:00-07:00"
```

Listing 4.3: Von Daten(rot/normal) getrennte Struktur (blau/fett).

Damit reduziert sich die Anzahl der zu übertragenen Zeichen von 107 auf 62. Die Nachricht besteht noch immer aus einer einzelnen Zeichenkette. Jedes Zeichen wird durch ein Byte kodiert. Der zweite Wert ist jedoch eine Prozentangabe, der

[3] JSON - JavaScript Object Notation wird spezifiziert in [Tbra14]

dritte Wert eine Zeit- und Datumsgabe. Angenommen für die Prozentangabe ist eine ganzzahlige Auflösung ausreichend, dann kann diese durch ein Byte dargestellt werden [0-255]. Für die Kodierung einer Datums- und Zeitangabe reichen nach *Unixzeit* 32 Bit und somit 4 Byte aus. In der Summe könnte die dargestellte Zeichenkette durch Typisierung auf 30 Byte Payload reduziert werden.

```
[Byte]RelativeHumidity:percent[Byte][Integer]
```

Listing 4.4: Typisierung der Daten. Erstes Byte gibt Länge des Strings an (max 255).

Da im ACDSense-Szenario die Zeichenkette am Anfang keine beliebige ist, sondern einen Sensortyp aus einer endlichen Menge benennt, kann der String durch einen komplexen Datentyp ersetzt werden. Die Anzahl der Sensortypen einer Wetterstation liegt deutlich unter 255, sodass ein Byte ausreicht. Schlussendlich bleiben 6 Byte zu übertragen. Was folglich noch sichergestellt werden muss, ist die Zuordnung von Daten und Struktur. An die Nachricht wird eine ID ergänzt, welche die Struktur identifiziert. Eine ID aus 4 Bytes als Integer kodiert könnte 4.294.967.295 Strukturen unterschieden. Die übertragende Nachricht hätte damit eine Größe von 10 Bytes (ehemals 107 Bytes). Die passende Pseudostruktur sowie die zugehörigen Kodierung der Beispielnachricht lauten:

```
{
     "struct-id":301,
     "sensorevent":
     {
          "type":
          {
               "type":byte,
               "map":{[0, temperature],[1, humidity],…}
          },
          "values":byte,
          "timestamp":int
     }
}

[301][1][49][1125012900]
```

Listing 4.5: Pseudostruktur (blau/fett) und entsprechende Kodierung (rot/normal) von Listing 4.2.

Dieses Vorgehen ist auf nahezu alle periodischen Daten gleicher Struktur anwendbar. Vorausgesetzt wird, dass Sender und Empfänger Zugang zur Struktur haben, zum Beispiel über ein in die Middleware integriertes Directory, sowie über Software verfügen, Struktur und Daten entsprechend zu kodieren und zu dekodieren.

5 KONZEPT UND IMPLEMENTATION

5.1 DISCOVERY MIT MQTT

In einem verteilten Kommunikationsnetz gibt es eine Vielzahl von Knoten, die Daten miteinander austauschen. Diese Strukturen können sich dynamisch verändern. Für eine zuverlässige und unkomplizierte Kommunikation benötigt es Mechanismen, um Teilnehmer in einem Netzwerk und deren bereitgestellte Dienste zu erfassen, oft unter dem Begriff Discovery zusammengeführt. Kapitel * unterscheidet dabei zwischen dem Entdecken von Knoten, sprich Geräten im Netzwerk, den von diesen Knoten angebotenen Diensten, und Ressourcen, welche oft Entitäten der Anwendung sind.

Im Falle von Mqtt sind Knoten sowohl Teilnehmer, welche sich mit zentralen Brokern verbinden, sowie die Broker selbst. Der Dienst ist das Wissen darüber, dass ein Netzwerkteilnehmer eindeutig als Mqtt-Broker fungiert, sowie alle vom Broker bereitgestellten Dienste auf Anwendungebene. Ressourcen sind primär die auf dem Broker vorhanden Topics. Ferner gehören Metainformationen über Publisher, Subscriber und dem Broker selbst dazu. Mqtt spezifziert allerdings keine der genannten Discoveryvarianten. Es ist also standardmäßig nicht möglich andere Mqtt-Teilnehmer im Netzwerk zu ermitteln, noch registrierte Topics auf dem Broker anzufragen. Es ist den Broker- und Bibltiotheksentwicklern überlassen, eigene Discoveryroutinen bereit zu stellen. Dadurch wird allerdings die Interoperabilität zwischen unterschiedlichen Implementationen gefährdet. Die Gründe, warum Mqtt diese Mechanismen nicht mit spezifiziert liegen zum einen im Designansatz, ein besonders leichtgewichtiges Protokoll bereit zu stellen, zum anderen in der Nutzung von TCP als Transportprotokoll, das kein IP-Multicast und damit keine Broadcastnachrichten im Netzwerk unterstützt. Des Weiteren ist Mqtt Daten- beziehungsweise Topic-zentriert. Es gibt keine logischen Ende zu Ende-Verbindungen und damit keine Relationen zwischen den Clients sondern zwischen Client und Server/Topic. Welche Möglichkeiten gibt es nun, auf anderem Wege eine Discovery mit Mqtt umzusetzen.

5.1.1 SERVICE DISCOVERY

Auf Netzwerkebene können von Mqtt unabhägige Discoveryprotokolle verwendet werden. Diese dienen dazu Knoten im Netzwerk zu finden und als Mqtt-Broker zu erkennen. Ein Beispiel hierfür sind Zeroconf-Techniken wie *Multicast-DNS* und *DNS-SD* (vgl. Kapitel *). Eine weitere Möglichkeit bestehe darin den Mqtt-Broker bei einer *DHCP* Registrierung konkret als solchen auszuweisen. Möchte man den Suchradius über lokale Netze hinaus ausweiten, müssen gegebenenfalls Erweiterungen implementiert werden. Da die Nachrichtenverteilung über einen Broker stattfindet, ist es überflüssig nach Mqtt-Clients zu suchen. Listing 5.1 zeigt eine Konfigurationsdatei, um einen Mqtt-Broker im Netzwerk über den mDNS/DNS-SD-Dienst *Avahi* zu registrieren.

```
<!-- Put this in /etc/avahi/services/mosquitto.service -->

<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
 <name replace-wildcards="yes">Mosquitto MQTT server on %h</name>
  <service>
   <type>_mqtt._tcp</type>
   <port>1883</port>
   <txt-record>info=Publish, Publish! Read all about it! mqtt.org</txt-record>
  </service>
</service-group>
```

Listing 5.1: Konfigurations-XML um eine Mqtt-Broker im Netzwerk mit Avahi zu registrieren [00i].

5.1.2 Ressource Discovery

Für das Auffinden des Dienstes selbst, stehen Protokoll-unabhängige Verfahren zur Verfügung. Anders verhält es sich bei der Erfassung von Mqtt-Ressourcen, insbesondere von Topics. Weder sieht die Spezifikation eine Möglichkeit vor Topics abzufragen, noch implementieren die in der Arbeit verwendeten Broker Mosquitto und Apollo eigene Ansätze einer Discovery. Die Notwendigkeit einer solchen Funktion zeigt sich im vorgestellten ACDSence (vgl. Kapitel*). Wetterstation senden periodisch Daten, konkret für diesen Fall, an einen Mqtt-Broker. Es gibt keine festgelegte Empfängerrelation. Ganz unterschiedlich können interessierte Instanzen sich für Wetterstationen und deren Daten registrieren. Dafür ist das Wissen über registrierte Wetterstationen und den ihnen zugeordneten Topics notwendig. Da dieses Wissen nicht vom Broker bereit gestellt wird, muss es einen zentralen Abfragedienst geben, der die Wetterstation-Topic-Relation auflöst oder eine vorher festgelegte statische Konfiguration, die jedem Empfänger zugänglich gemacht wird. Damit erhöhen sich die Verwaltungskosten und die Fehleranfälligkeit. Im Folgenden soll daher eine Variante vorgestellt werden, die ohne separate Dienste auskommt und Discoveryanfragen unabhängig von der Implementation über den Mqtt-Broker verarbeitet.

5.1.3 Topic Discovery

Die einzige Möglichkeit mit einem Mqtt-Broker zu kommunizieren, ist es sich für Topics zu registrieren und Nachrichten an Topics zu versenden. Eine Discovery muss demnach selber ein kommunizierender Client sein und mit diesen Relationen auskommen. Die einfachste Form einer Discovery wäre ein Client, der sich für alle Topics registriert. Der entsprechende Ausdruck wäre:

```
#       // Subscription für alle Topics
```

Ein Empfänger würde dann alle Nachrichten erhalten, welche auf dem Broker eintreffen. Bei vielen tausend aktiven Verbindungen wären die Last und der Anteil nicht relevanter Informationen enorm. Wenn alle Teilnehmer eine solche Methode implementieren, würden sich obendrein die Geschwindigkeit und Stabilität des Brokers zunehmend verschlechtern. Es ist daher eine sinnvolle Strukturierung notwendig.

Eine Möglichkeit ist ein globales Discovery-Topic auf dem Broker einzurichten. Darin können alle Sender jene Topics veröffentlichen, welche für andere Clients sichtbar sein sollen. Dabei muss berücksichtigt werden, dass es keine doppelte Belegung von veröffentlichten Topics gibt. Anders gesagt muss der Veröffentlicher

darauf hingewiesen werden, wenn ein entsprechendes Topic bereits registriert wurde. Ferner ist zu verhindern, dass Topics von anderen überschrieben oder gelöscht werden. Der Discovery-Client würde sich für ein verbindliches Discovery - Topic anmelden.

```
DISCOVERY/#  // Subscription für ein verbindliches Discovery-
Topic
```

Ein Client erhält nach wie vor alle auffindbaren Topics, aber eben nur jene, die vorher explizit registriert wurden. Ein Sender muss daher sein Topic aktiv veröffentlichen. Ein Empfänger muss von der gesamten Menge an Topic, die für ihn relevanten filtern.

Um diese Nachteile zu unterbinden, wird das Konzept einer *Domäne* eingeführt. Im Grunde handelt es sich dabei um ein Topic, das für alle Untertopics eine Art Container bildet. Mqtt lässt dies dank seiner hierarchischen Topicverarbeitung zu. Ein solcher Container kann repräsentativ für eine Anwendung stehen. Alle von der Anwendung erstellten Topics werden innerhalb der Topicdomäne strukturiert. Angenommen eine Firma benachrichtigt ihre Fahrer über die Topics *Fahrer1* und *Fahrer2*, dann könnte die Domäne *firma* sein und die verwendeten Topics *firma/fahrer1* und *firma/fahrer2*. In Kombination mit dem vorherigen Ansatz lautet die Subscription für die Discovery:

```
// Subscription für eine Discovery innerhalb einer Domäne
firma/DISCOVERY/#
```

Dadurch erhält ein Empfänger nun bei einer Discovery einzig die innerhalb einer Domäne veröffentlichten Topics, womit sich der Nachrichtenaufwand signifikant reduziert. Desweiteren müssen Sender ihre Topics nicht explizit veröffentlichen. Mit Hilfe einer beim Sender aktiven Discovery können innerhalb der Domäne angelegt Topics erkannt und automatisch im Discovery-Topic registriert werden. Der generelle Ablauf ist in Tabelle 5.1 dargestellt.

Für alle vorgestellten Lösungen gilt, dass bei der Registrierung des Topics die Option *retained* aktiviert ist, damit Nachrichten persistent gehalten und ausgeliefert werden, wenn eine Discovery-Subscription erst nach der Topic-Registrierung erfolgt. Des Weiteren ist eine zuverlässige Übertragung notwendig, um die Registrierung und das Empfangen aller Topics zu garantieren.

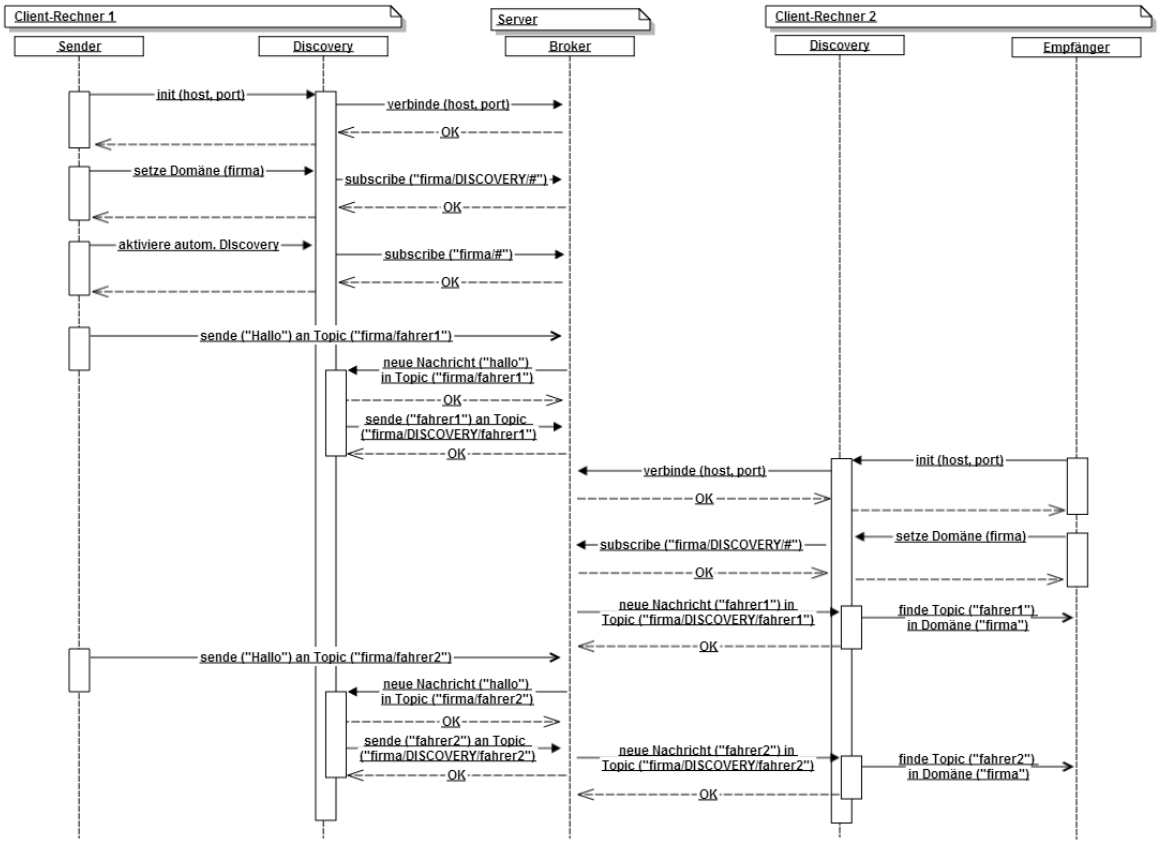


Tabelle 5.1: Ablauf einer Discovery nach dem Domänenmodell. Sender und Empfänger initialisieren jeweils ein Discovery-Objekt und legen eine Domäne fest. Daraufhin erfasst die Discovery neu erstellte Topics in der Domäne und reicht diese Information an die Clients weiter.

5.2 NACHRICHTENIDENTIFIZIERUNG

Die Simulationsumgebung aus dem ACDSense-Szenario protokolliert im Kern jedes Sende- und Empfangsereignis. Genauer gesagt werden Zeitpunkt, Sender, Empfänger, Payload- und Nachrichtengröße, sowie die Nachrichten-ID erfasst. Im Falle von XMPP und SMACK ist diese eindeutig. Das muss sie sein, um bei der Auswertung die Latenz für die Übertragung einer Nachricht errechnen zu können. MQTT unterstützt keine systemweiten eindeutigen Nachrichten-IDs. Nachrichten können nur innerhalb einer Client-Server Sitzung voneinander unterschieden werden. Es ist daher eine Methode notwendig, um Nachrichten systemweit und damit über Rechnergrenzen hinweg, eindeutig zu identifizieren.

Um das spätere Testergebnis nicht zu beeinflussen, ist zu Übertragung von zusätzlichen Informationen zu vermeiden. Dadurch erübrigt sich die Integration einer ID im Payload. Der Payload ist andererseits das einzige, das Sender und Empfänger miteinander austauschen. Denn logische Relationen bestehen bei MQTT zwischen Sender und Topic, respektive Topic und Empfänger. Damit ist eine Platzierung einer ID im Header ebenso hinfällig, denn Headerinformationen werden nur vom Server verarbeitet. Im Szenario stehen Topics für Wetterstationen, genauer gesagt für die Informationen, die eine Wetterstation veröffentlich. Eine Topic kann daher selber nicht eindeutig sein. Für die Methode gilt letztendlich: Sie muss aus dem Payload einer Nachricht eine eindeutige ID generieren können, ohne den Payload selber zu verändern.

Die einfachste Lösung ist, den Payload selber als ID zu nutzen. ACDSense veröffentlicht Wetterdaten als String. Zeichenketten nehmen im Vergleich zu anderen Datentypen viel Speicherplatz ein. Mit der Aufzeichnung jeder Nachricht und damit jeder ID wirken sich IDs von Typ String signifikant auf den Bedarf an flüchtigen und persistenten Speicher aus. Im Grunde handelt es sich um die Verdopplung des Payloads. Ferner ist die Payloadlänge variabel. Damit wäre die Länge einer ID ebenso variabel. Sinnvoller wären IDs gleicher Länge in einem kompakteren Format.

Dafür eigenen sich Hashfunktionen, die Objekte unterschiedlicher Länge aus einer Menge auf eine Menge mit Objekten gleicher Länge abbildet. Eine solche Funktion ist *Message-Digest 5* (MD5). MD5 erzeugt bei der Abbildung Hashwerte aus 128 Bit, sprich 16 Bytes. Hashfunktionen sind in der Regel nicht injektiv. Das heißt, es ist möglich, dass zwei Objekte aus der Quellmenge auf den gleichen Hashwert abgebildet werden. In der Tat ist der MD5-Algorithmus nicht eindeutig *. In zahlreichen Testläufen sind jedoch keine Kollisionen aufgetreten, sodass die Eindeutigkeit für das gegenwärtige Szenario als ausreichend eingestuft wird.

Für einen nahezu eindeutigen Hashwert, sollte der Quellwert eindeutig sein. Der Quellwert ist, wie bereits erwähnt, der Payload und damit die Information einer Wetterstation. Werfen wir einen Blick auf eine typische Nachricht.

```
{"sensorevent":{"type":"AirTemperature:fahrenheit","values":[
77.0],"timestamp":"2005-08-26T00:00:00-07:00"}}
```

Eine Nachricht besteht demnach aus einem Ereignistyp (`sensorevent`), aus einem Wertetyp (`AirTemperatur:fahrenheit`), den zugehörigen Werten (`77.0`) und einem Zeitstempel (`2005-08-26T00:00:00-07:00`). In der Annahme, dass mehrere Wetterstationen im System registriert sind, welche mit gleichen oder ähnlichen Sensoren ausgestattet sind, ist die Wahrscheinlichkeit hoch, dass eine Nachricht dieser Art mehr als einmal im System auftauchen kann. Die Nacht selber ist daher nicht eindeutig. Lediglich das Topic, in dem eine Nachricht veröffentlich wird, steht zugleich als Information auf Sender- und Empfängerseite zur Verfügung. Die Kombination aus Topic und Nachricht ist indes eindeutig, denn Topics sind nach Wetterstationen benannt und eine Wetterstation, respektive deren Benennung ist systemweit singulär. Voraussetzung für eine eindeutige Kombination ist, dass kein Sensor einer Wetterstation zum gleichen Zeitpunkt mehr als eine Nachricht sendet. Schlussendlich wird für MQTT die Nachrichten-ID über einen MD5-Hash aus der Kombination von Topic und Nachricht gebildet. Die Umsetzung von MD5 in Java verlangt ein Bytearray als Eingabe. Dieses Bytearray ist demensprechend eine Konkatenation der Byterepräsentationen von Topic und Payload.

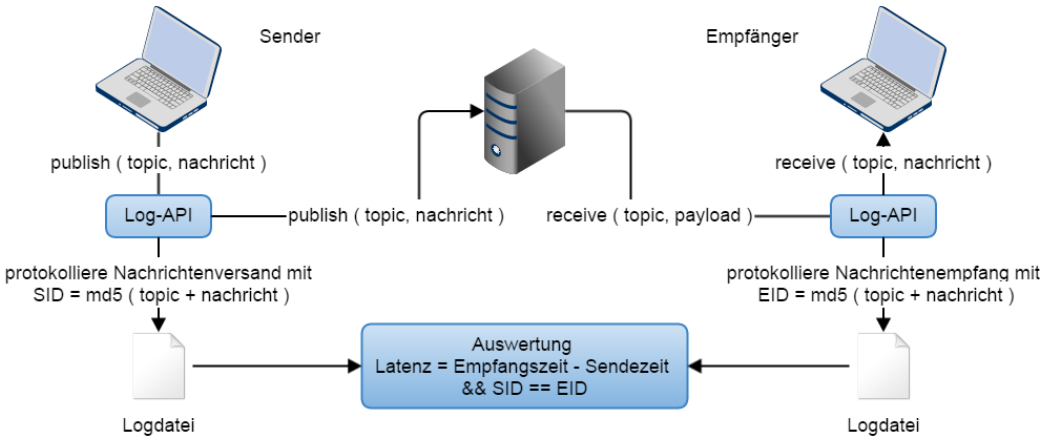


Tabelle 5.2: Generierung eines MD5-Hashwertes aus Topic und Payload zur Identifizierung einer MQTT-Nachricht für die spätere Auswertung.

6 TESTUMFELD

Bevor mit dem eigentlichen Testen begonnen werden kann, müssen Aufbau, Anforderungen und Kriterien spezifiziert werden.

6.1 AUFBAU

Lokaler Test

Bei einem lokalen Test sind alle beteiligten Testinstanzen (Rechner) in einem lokalen Netzwerk über Netzwerkkabel verbunden. Idealerweise handelt es sich dabei um ein LAN. Die Kommunikation findet ausschließlich innerhalb des LANs statt.

Entfernter Test

Mindestens eine Instanz ist nicht innerhalb eines lokalen Netzwerks platziert und über das Internet erreichbar. Daher ergeben sich Kommunikationsströme, die übers Internet geleitet werden.

Drahtgebunden

Bedeutet, dass alle Rechner per Ethernetkabel miteinander, respektive über einen oder mehrere Netzwerk-Switches verbunden sind.

Drahtlos

Mindestens eine Instanz nutzt als Netzwerkschnittstelle Funktechniken, wie WLAN oder Mobilfunk.

Einfacher Testaufbau

Bei einem einfachen Testaubau werden die Testläufe auf einem einzelnen Rechner ausgeführt. Alle Sender und Empfänger von Nachrichten befinden sich daher auf derselben Maschine. Einzig notwendige Server und Broker werden auf einem weiteren Rechner isoliert betrieben. Man bezeichnet einen solchen Aufbau auch als *Loopback* oder *Schleifenschaltung*.

Kollaborativer Testaufbau

In dieser Arbeit handelt es sich um einen kollaborativen Testaufbau, wenn sich Kommunikationsteilnehmer, Sender und Empfänger auf unterschiedlichen Rechnern befinden. Damit ist nicht eine mögliche Kollaboration zwischen Servern oder Brokern gemeint (http://xmpp.org/extensions/xep-0238.html, http://mqtt.org/wiki/doku.php/bridge_protocol). *

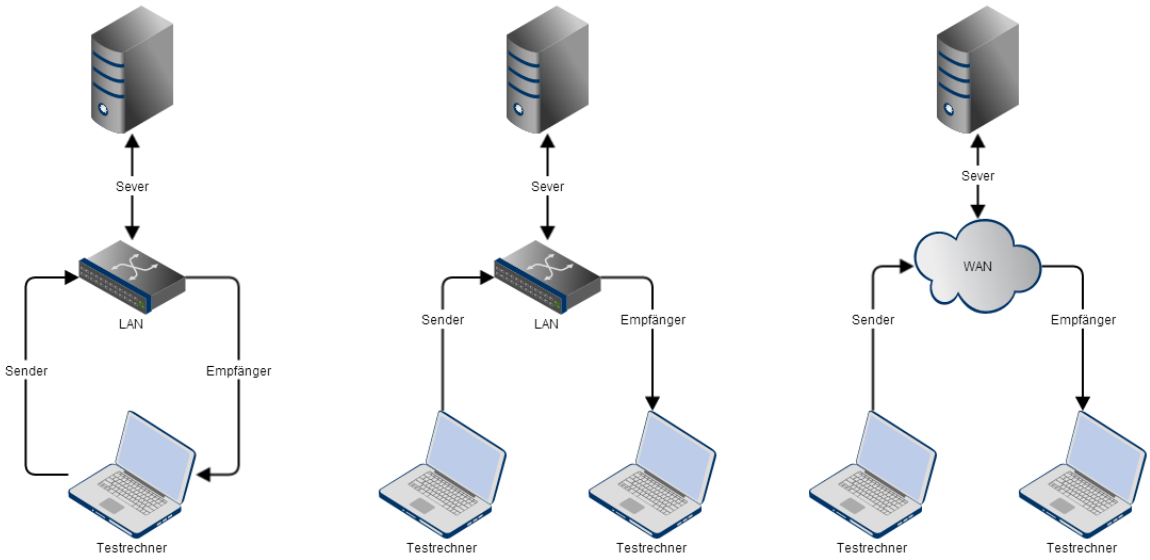


Tabelle 6.1: Testaubauten: Links – lokaler einfacher Test, Mitte – lokaler kollaborativer Test, Rechts – entfernter kollaborativer Test

6.2 ANFORDERUNGEN

Perfomancemessungen sind aus verschiedenen Gründen nützlich und notwendig. Zum einen erlauben sie Aussagen über die Leistung des Systems und machen es so vergleichbar mit anderen bestehenden Implementationen. Im Idealfall wird dadurch ein Mehrwert deutlich, wodurch die Markposition der Software verbessert wird. Zum anderen benötigt man ein Diagnosewerkzeug, um auftretende Leistungsdefizite während der Entwicklung zu identifizieren.

Durch Messungen können Aussagen getroffen werden über, zum Beispiel, die Anzahl von Nachrichten, die innerhalb einer Sekunde verarbeiten/versendet werden können oder wie hoch die durchschnittliche Latenz für eine Nachrichtenübertragung ist. Für die Bewertung eines Systems sind detailliertere Informationen notwendig. Dafür betrachtet man die Wertespitzen, wann und wie oft diese Auftreten. Ein System mag in 9 von 10 Fällen hervorragende Übertragungseigenschaften aufweisen, eine extreme Abweichung nach oben kann es für sicherheits- und zeitkritische Anwendungen ungeeignet werden lassen.

Es ist wichtig bei der Identifikation von Spitzen den Zeitpunkt und "Ort" in die Betrachtung mit aufzunehmen. Treten diese am Anfang, am Ende oder bei Lastzunahme auf? Das Interesse beschränkt sich daher nicht nur auf aggregierte Werte, sondern berücksichtigt den Verlauf von Latenz und Durchsatz als Ganzes.

Die in der Arbeit genutzten Systeme sind zeitlich diskret. Nachrichten verlassen oder erreichen ein System zu einem bestimmten Zeitpunkt. Alle daraus resultierenden Messkurven sind von Prinzip aus unstetig. Es ist nicht möglich die Latenz oder den Durchsatz für einen exakten Zeitpunkt zu bestimmen. Jedoch ist es möglich genannte Parameter für eine Zeitspanne oder für eine Nachricht oder Nachrichten zu ermitteln.

Reduzierung von Messmanipulationen

Kriterien einer idealen Testumgebung sind:

- Ein dedizierter Prozessor/Kern für den Sender um Einflüsse anderer Prozesse zu vermeiden.
- Ein dedizierter Prozessor/Kern für den Broker/Server um Einflüsse anderer Prozesse zu vermeiden.
- Ein dedizierter Prozessor/Kern für den Empfänger um Einflüsse anderer Prozesse zu vermeiden.
- Ein separates Netzwerk um Einflüsse anderer Übertragungsströme zu vermeiden
- Arbeitsspeicher für erfasste Messwerte allokieren, anstatt paralleler aufwendiger Schreibeoperationen.
- Für Sender und Empfänger voneinander getrennte Verbindungen verwenden und in separaten Threads ausführen.
- Statistische Auswertungen erst nach dem Test ausführen.

Messparameter

N	Anzahl Nachrichten
S_n	Sendezeitpunkt von Nachricht n
E_n	Empfangszeitpunk von Nachricht n
L_n	Übertragungslatenz von Nachricht n
SO_n	Sende-Overhead von Nachricht n
EO_n	Empfangs-Overhead von Nachricht n
SR_n	Durchsatz zum Zeitpunkt von n gesendeten Nachrichten (Nachrichten/Sekunde)
ER_n	Durchsatz zum Zeitpunkt von n empfangenen Nachrichten (Nachrichten/Sekunde)

Herleitungen

Die *Latenz* einer Nachricht n berechnet sich aus dem Zeitintervall zwischen Versand und Empfang der Nachricht n

$$Latenz = L_n = E_n - S_n$$

Der *Durchsatz*, die Anzahl an Nachrichten innerhalb eines Zeitfensters, lässt sich Errechnen mit

$$Durchsatz = \frac{E_n - S_1}{N}$$

Dieser Ausdruck ist für niedrige N ungeeignet, da er für niedrige Werte weniger eine Beschreibung für den Durchsatz, sondern für die Latenz darstellt. Bei $N = 1$ ist er sogar gleich der Latenz. Ferner ist es schwierig die Sende- und Empfangsrate in einer einzelnen Metrik konsistent darzustellen. Daher betrachtet man im Folgenden beide Raten getrennt.

$$Senderate = SR_n = \frac{1s}{SO_n}$$

$$Empfangsrate = ER_n = \frac{1s}{EO_n}$$

Die eigentlichen Werte für Senderate und Empfangsrate beschreiben die Anzahl gesendeter Nachrichten, beziehungsweise die Anzahl empfangener Nachrichten innerhalb eines Zeitintervalls. Für die Berechnung benötigt man außerdem die Werte für den Sende- und Empfangsoverhead, welche die benötigte Zeit zum Senden, zwischen Senden und Empfangen, zwischen Empfangen einer Nachricht angeben.

$$Sendeoverhead = SO_n = S_n - S_{n-1}$$

$$Empfangsoverhead = EO_n = E_n - E_{n-1}$$

Damit können nun Senderate und Empfangsrate bestimmt werden. Beide Raten geben den Durchsatz für eine bestimmte Nachricht an und nicht für einen konkreten Zeitpunkt. Die bisherige Lösung zeigt Schwächen bei hohen Durchsätzen. Ein Beispiel: ist $SO_n = 1ms,\ dann\ ist\ SR_n = 1000,$ ist $SO_n = 2ms,$ $dann\ ist\ SR_n = 500.$ Wie oben erwähnt, ist es oftmals nur möglich in einer Auflösung von Millisekunden zu messen, was zu solchen Wertsprüngen führt. Eine Messung in Nanosekunden würde genügend gute Ergebnisse liefern, ist jedoch nicht überall gegeben. Daher mindert man das Problem, indem man den Durchsatz nicht für eine Nachricht, sondern für eine Zahl M von Nachrichten angibt.

$$SO_n = \frac{S_n - S_{n-M}}{M}$$

$$EO_n = \frac{E_n - E_{n-M}}{M}$$

Für die obige Formel gilt $M = 1$. Beachtet werden muss, dass für $N \leq M$ der Durchsatz undefiniert ist.

Zeit-Synchronisation

Für eine Vielzahl der Messungen ist es essentiell, den Zeitpunkt von Ereignissen zu erfassen und zwar genau in dem Moment, in den das Ereignis eintritt. Der Empfang einer Nachricht wäre ein solches Ereignis. Irgendwo in der damit verbundenen Abarbeitungskette im Programm muss der Zeitpunkt registriert werden. In der Regel findet die Zeitmessung im Quellcode kurz vor oder kurz nach einer bestimmten Operation statt um die Abweichung gering zu halten. Zwischen Ereignis und Messung dürfen daher keine oder keine aufwendigen Berechnungen liegen. Beachtet man dies, bleibt die Zeitabweichung sehr gering, ist aber vorhanden und sollte im Bewusstsein bleiben. Ferner ist es von Programmiersprache, Laufzeitumgebung und Betriebssystem abhängig, wie präzise und in welcher Auflösung eine Zeiterfassung möglich ist.

Betriebssystem	Auflösung
Windows 95/98	**55 ms**
Windows NT, 2000, XP – Einkern-Prozessor	**10 ms**
Windows XP - Mehrkern-Prozessor	**15.625 ms**
Windows 7	**1 ms**
Linux 2.4 kernel	**10 ms**
Linux 2.6 kernel	**1 ms**

Tabelle 6.2: Genauigkeit der Zeitgeber von Betriebssystemen nach [08, 13].

Für die Zeitmessung in Java stellt die Standardbibliothek zwei Methoden zur Zeiterfassung zur Verfügung. Die erste liefert Ergebnisse in Millisekunden. Die Präzession ist allerdings abhängig vom Betriebssystem und liegt bei Windows XP im Bereich von circa 16 Millisekunden. Letztere ermöglicht Messungen im

Nanosekundenbereich. Deren Genauigkeit ist ebenso abhängig vom Betriebssystem. Der Rechenaufwand ist derweil deutlich höher und die Methode ist weniger geeignet für eine Zeitsynchronisation und wird daher für lokale und relative Laufzeitmessungen verwendet.

Für die Testumgebung ergibt sich folgendes: Alle Ereignisse müssen beim Eintreten registriert und mit einer Zeitinformation gekoppelt werden. Diese Zeitinformation wird über eine zur Verfügung stehende Schnittstelle abgerufen. Deren Genauigkeit und Auflösung ist abhängig vom Betriebssystem und damit von der Systemzeit. Die Systemzeit ist auf getrennten Rechnern unterschiedlich. Das Auftreten desselben Ereignisses kann daher zu verschiedenen Zeitpunkten gemessen werden. Diese Differenz ist damit ein Messfehler. Dessen Einfluss ist abhängig von der relativen Zeitverschiebung auf den beteiligten Rechnern. Es muss daher versucht werden, alle Zeitgeber miteinander zu synchronisieren, um die relative Verschiebung gering zu halten.

Betriebssysteme gleichen sich in der Regel automatisch mit Zeitservern im Internet ab. Nicht selten verwenden sie dazu das *Network Time Protocol*. Dafür steht eine Anzahl von Rechnern im Internet bereit, welche als Zeitgeber für andere Rechner fungieren. Client und Server tauschen UDP-Pakete miteinander aus, um mit Hilfe der jeweiligen Server-Zeit und den Übertragungslatenzen die jeweiligen Uhren abzugleichen. Dabei gilt, je mehr Server angesprochen, umso genauer, je länger und langsamer die Kommunikationswege, desto ungenauer ist die Synchronisation schlussendlich. Beide Kriterien beeinflussen sich gegenseitig.

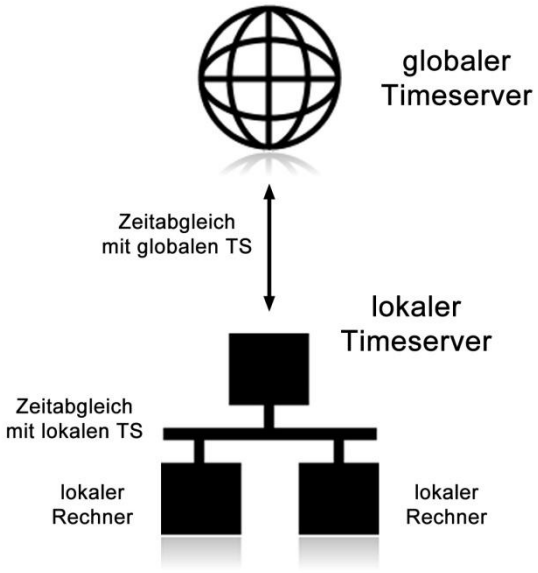


Tabelle 6.3: Zeitsynchronisierung mit NTP im LAN.

Die Messungen im *einfachen Testfall* finden lediglich auf einem Rechner statt und nutzen daher dieselbe Zeitquelle. Eintreffende und Gesendete Paket werden durch die Systemuhr erfasst. Eine Synchronisation nach außen ist nicht notwendig. Im *internen kollaborativen Testfall* kommunizieren unterschiedliche Rechner mit verschieden Systemuhren und müssen daher synchronisiert werden. Die Rechner befinden sich in einem lokalen Netzwerk. Die Übertragungswege und damit die Latenzen sind gering. Ein Rechner im Netzwerk wird als Zeitgeber gewählt, der, falls gewünscht, seine Systemzeit mit dem Internet abgleicht. Alle anderen Rechner im Netzwerk gleichen ihre Uhren mit dem lokalen Zeitserver ab (vgl. Tabelle 6.3). Der *externe kollaborative Testfall* ist gleichwohl schwieriger zu handhaben, da die Wege zwischen den einzelnen kommunizierenden Rechner

selten vorhersehbar sind. Es ist ratsam einige Testreihen durchzuführen, um zu ermitteln ob eine direkte Synchronisation untereinander, ähnlich dem internen Fall, oder eine jeweilige Synchronisation mit einem Zeitserver im Internet, bessere Ergebnisse liefert. Da die Uhren der globalen Zeitserver auch untereinander abweichen, sollte derselbe Server für alle Rechner als Zeitgeber dienen.

Daten-Synchronisation

Es ist abzusehen, dass für die späteren Tests eine Reihe von Daten zwischen den beteiligten Rechnern verteilt werden müssen. Zum Einen die Platzierung des Testanwendungen selbst um Sender und Empfänger auf den entsprechendem Gerät erzeugen zu können. Zum Andere benötigt jede Applikation eine Konfigurationsdatei, welche die jeweiligen Parameter, wie Sender-/Empfängeranzahl oder das zu testende Protokoll angibt. Neben den Konfigurationsdateien werden Testressourcen benötigt. Auf der Senderseite müssen die protokollierten Wetterdaten zu Verfügung stehen. Nicht zuletzt erzeugen die Testapplikationen während ihrer Ausführung Protokolle über den Nachrichtenverlauf. Diese Protokolle werden später bei der Auswertung wieder an zentraler Stelle benötigt.

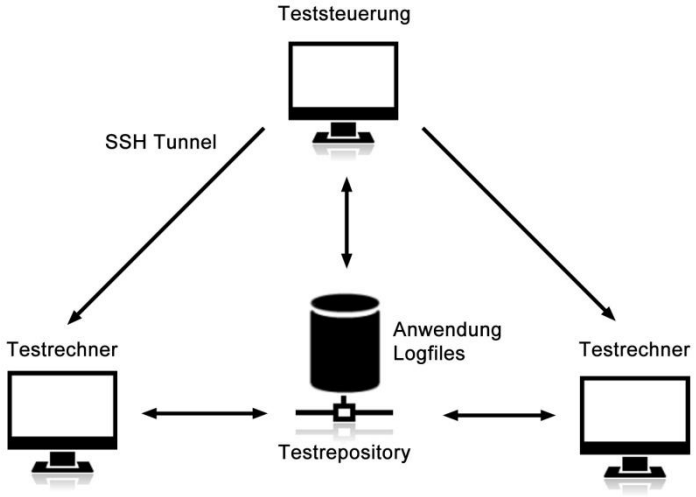


Tabelle 6.4: Datensynchronisierung entfernter Rechner.

Eine einfache Möglichkeit die Daten auf allen Rechnern synchron zu halten sind Versionsverwaltungssystem. Über ein zentral zugängliches Repository werden die benötigten Daten synchronisiert. Es ist nicht nötigt eine aktualisierte Testapplikation auf jeden Rechner einzeln zu überspielen. Die Synchronisierung des Repository. Ähnlich verhält es sich mit den aufgezeichneten Testprotokollen. Eine Synchronisierung nach dem Testdurchlauf ermöglicht die spätere Auswertung vom Repository aus. Natürlich sollte die Verzeichnisstruktur sinnvoll gewählt werden Konflikte zu vermeiden. Eine sinnvolle Erweiterung ist eine entfernte Steuerung der Testrechner. Über SSH können sichere Netzwerkverbindungen zu entfernten Geräten hergestellt werden, um die lokale Kommandozeile verfügbar zu machen. Damit ist die Steuerung der entfernten Testapplikationen von nur einem Rechner aus möglich (vgl. Tabelle 6.4).

6.3 AUSWERTUNG

Diagramme

Die spätere Auswertung konzentriert sich vor allem auf die Übertragungslatenz von Nachrichten. Entsprechende Visualisierungen stellen entweder die Latenz in Abhängigkeit zur Nachricht oder zu einem Zeitpunkt (Empfangszeit/Sendezeit) dar. Der Vorteil der ersten Variante ist, man erkennt für jede einzelne Nachricht deren Übertragungslatenz. Der Vorteil der Zweiten ist, man erkennt das Nachrichtenaufkommen über die Zeit hinweg. Eine übliche Visualisierung sind Liniendiagramme. Liniendiagramme haben allerdings einen entscheidenden Nachteil. Auf der Ordinate kann immer nur ein Wert verzeichnet werden. Nimmt man die Abzisse zur Darstellung von Nachrichten, entstehen Problem Beschränkungen in Fällen mit einem Sender und vielen Empfängern, denn für die gesendeten Nachrichten gibt es exakt so viele gemessene Latenzen, wie es Empfänger gibt. Nimmt man die Abszisse zur Darstellung eines Zeitpunkts ergebenen sich Konflikte für Ereignisse, die zum gleichen Zeitpunkt auftreten. Es sollte daher eine Visualisierung gewählt werden, die für einen Argument mehrere Ergebnisse darstellen kann. Eine solche Möglichkeit sind Streudiagramme. Streudiagramme repräsentieren jedes Wertepaar als Punkt. Tabelle 6.5 zeigt die beiden Visualisierungsarten im Vergleich.

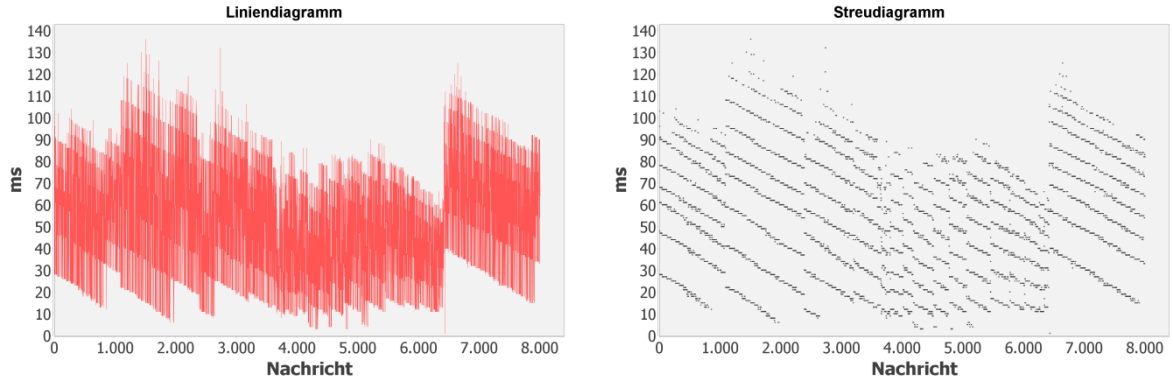


Tabelle 6.5: Darstellung von Messwerten mit Liniendiagramm und Streudiagramm.

Knime

Die Auswertung der Testprotokolle erfolgt mit KNIME. KNIME ist eine offene datengetriebene Analyseplattform und unterstützt den Arbeitsprozess von der Datenerfassung,- aufarbeitung, -transformation, -analyse bis hin zur Visualisierung und Vverwertung. Technisch basiert KNIME auf Eclipse, ist modular und erweiterbar. KNIME integriert Java, R und JChart direkt in den Arbeitsprozess. Arbeitsschritte in KNIME werden in Knoten gekapselt und in Pipelines angeordnet.

Im Folgenden soll Beispielhaft die Auswertung von ACDSense-Daten skizziert werden. Der dazugehörige KNIME-Workflow ist im Anhang zu finden.

1. Zu Beginn erfolgt das Einlesen der Protokolle. Für die Latenzberechnung benötigen diese vom Sender und Empfänger. Da für jeden Sender/Empfänger ein separates Protokoll aufgesetzt wird, müssen diese entsprechend aggregiert werden
2. Noch vor der Zusammenführung von Sender- und Empfängerprotokollen, können für beide Seiten die Sende- und Empfangsrate berechnet werden.
3. Die Kombination beider Seiten erfolgt über die in den Protokollen registrierte Nachrichten-ID. Der resultierende Datensatz enthält jetzt für jede Nachrichten-ID den Sende- und Empfangszeitpunkt.

4. Aus der Differenz beider Werte lässt sich die Übertragungslatenz errechnen. Dieser Teilschritt ist in Tabelle 6.6 dargestellt.
5. Die erhaltenen Werte können nun auf verschiedene Art über *JChart*-Diagramme oder *R*-Skripte visualisiert werden.
6. Am Schluss folg die Speicherung der Berechnungen im CSV-Format und der Diagramme als Bilddateien.

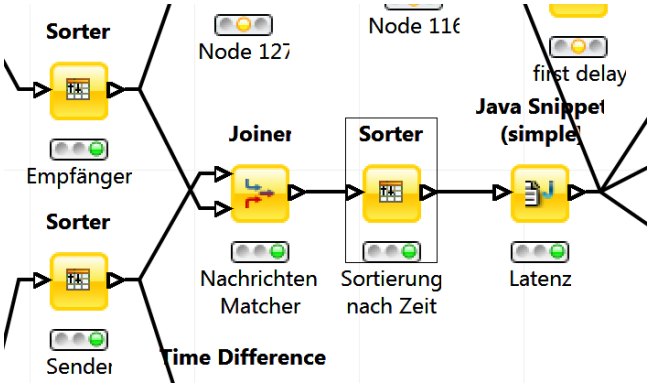


Tabelle 6.6: Teilschritt aus KNIME zur Auswertung von ACDSence-Experimenten. Sender- und Empfängeraufzeichnungen werden über die Nachrichten-ID kombiniert und nach der Zeit sortiert. Aus Sende- und Empfangszeit kann Latenz berechnet werden.

7 Evaluation

Übersicht

7.1 Synchronisation

Bisher liefen alle Testfälle auf einem Testrechner ab, der mit einem Broker oder Server im LAN verbunden war. Eine große Zahl von Sendern und Empfängern auf einer Maschine ist jedoch kein realistisches Szenario. Zweckmäßig ist es, Tests auf getrennten Rechnern auszulagern. Wie bereits diskutiert, müssen dafür die Systemuhren synchronisiert werden. Nach der analytischen Betrachtung soll untersucht werden, wie zuverlässig NTP in einer realen Testumgebung arbeitet.

Die Frage nach der Zuverlässigkeit lässt sich durch die Differenz ausrücken, die beide getrennte Uhren beschreibt. Je größer die Differenz, desto unzuverlässiger die Synchronisation. Da NTP eine kontinuierliche Synchronisation der Uhr mit einer zeitgebenden Uhr realisiert, ist auch die Zuverlässigkeit kontinuierlich zu bewerten. Die Zuverlässigkeit wird dann durch statistische Größen, wie Maximum und Standardabweichung der Menge von Zeitdifferenzen gebildet. Die gängige Methode ist daher den Zeitunterschied periodisch zu messen. Eine weitere Möglichkeit sind Referenzmessungen. Man wählt ein Szenario, das mit großer Wahrscheinlichkeit gleiche Ergebnisse bei gleichen Testparametern erzeugt. Das Szenario sollte daher endlich, idealerweise nicht komplex, terminierend und determinierend sein. In hochdynamischen Kommunikationsnetzen sind diese Eigenschaften nicht ohne weiteres zu erreichen. Das Szenario kann jedoch so gewählt werden, dass die genannten Anforderungen in hohem Maße zutreffen.

Es bietet sich ein Szenario zu wählen, bei dem es vorhersagbare und relativ konstant bleibende Werteentwicklungen gibt. Daher entscheiden wir uns gegen die Testreihe im ACDS-Project und den damit verbundenen variablen Übertragungsparametern, der Vielzahl von nebenläufigen Entitäten und zeitlich stark schwankenden Ereignisströmen. Eine Wiederholung von TC 1 ist besser geeignet. Darin zeigen sich konstant bleibende Perioden und von Natur aus keine Einflüsse auf Übertragungsparameter. Testfall TF# Verbindungsaufbau untersucht die Zeitspanne, die ein Client zum Aufbau eine Verbindung und die Zeitspanne für die Übertragung und Empfang einer Nachricht benötigt. Die jeweiligen Messungen werden n mal wiederholt. Sollte die Testreihe nach einer bestimmten Zeit nicht beendet sein, wird sie beendet. Die einzelnen Vorgänge, wie Verbindungsaufbau und Übertragung sind ebenfalls zeitgebunden und werden garantiert beendet. Es gibt einen Sender, einen Empfänger, einen Vermittler. Das Szenario ist endlich und terminiert. Innerhalb der Testreihen gibt es nur einen aktiven Sender und einen aktiven Empfänger auf dem Testrechner. Der entfernte Broker hält ebenso nur Verbindungen zu diesen zwei Instanzen. Die einzelnen Übertragungen laufen sequentiell mit genügend großer Zeitverzögerung ab. Das Träger-LAN ist stabil und ausreichend schnell. Es gibt keine weiteren signifikanten Verkehrsströme oder Lasterzeugnisse. Das Szenario ist daher nicht komplex. Die Frage nach der Determiniertheit ist die Frage nachdem erwarteten Ergebnis. Das Ergebnis ist die Übertragungslatenz, die Zeitdifferenz zwischen Senden und Empfang einer Nachricht. Würde bei jeder Messung die gleiche gemessene Latenz auftreten, wäre das Szenario determinierend. Im Realfall, wie auch in Testfall# zu beobachten, gibt es Schwankungen in den Latenzen, ebenso gelegentliche Ausreißer. Das Szenario ist also in einem bestimmten Maße

determinierend. Wie hoch dieses Maß ist, soll durch Wiederholung und Vergleich bestimmt werden.

- Unterschied Median vs. Mean
- Evaluation, Auswertung, Ausblick

Die folgenden Tests dienen als Vergleichsbasis. Zuerst wird die durchschnittliche Übertragungslatenz einer Nachricht auf einen Testrechner betrachtet. Das heißt Sender und Empfänger befinden sich auf einer Maschine. Später erfolgt eine Trennung von Sender und Empfänger auf verschiedenen Maschinen. Daher wird der erst genannte Test auf allen späteren Maschinen wiederholt. Dadurch soll erfasst werden, in wie weit ein Rechner die Ergebnisse beeinflusst. Bei dem Test mit getrennten Rechnern sind ein Sender auf der einen und ein Empfänger auf einer anderen Maschine installiert. Es erfolgt zuerst ein nicht synchronisierter Test, um die Differenz beider Rechneruhren zu erfassen. Danach werden die Uhren mit den in Kapitel * vorgestellten Mechanismus synchronisiert und der Test wiederholt. Die Testergebnisse sollten idealerweise identisch mit denen der lokalen Testumgebung sein. Die Abweichung beschreibt das Maß der Synchronisation.

7.1.1 SZENARIO

Das Testszenario ist wie folgt gestaltet. Auf den jeweiligen Rechnern wird ein Sender und eine Empfänger gestartet. Diese melden sich beim entsprechenden Vermittler an. Daraufhin beginnt der Test. Der Sender verschickt in periodischen Abständen eine eindeutig identifizierbare Nachricht, die vom Empfänger erhalten wird. Jeweils der Versand und der Empfang der Nachricht werden protokolliert. Die Differenz der beiden Werte ergibt die Übertragungslatenz. Es werden pro Test insgesamt 1000 Nachrichten versendet.

- Abbildung Testablauf

7.1.2 DURCHFÜHRUNG

Aufbau

Tabelle 7.16 fasst die unterschiedlichen Parameter für alle Test zusammen. Insgesamt gibt es vier Testvarianten, zwei lokale für die jeweiligen Rechner (*LokLnx*, *LokWin*) und zwei entfernte mit wechselnder Kommunikationsrichtung. Ein Rechner läuft mit Ubuntu, der andere mit Windows. Wechselnde Kommunikationsrechner bedeutet, dass einmal der Ubuntorechner als Sendeinstanz agiert und im folgenden Test als Empfänger (*EntWinLnx*, *EntLnxWin*). Kommuniziert wird über Mqtt. *Mosquitto* dient als Brokerimplementation. Die Nachrichten werden im Abstand von 500ms unzuverlässig und damit *best effort* übertragen (QoS 0). Die lokalen Tests werden dreimal wiederholt, die kollaborativen jeweils einmal. Alle Instanzen sind über das lokale drahtgebundene Institutsnetzwerk verbunden.

Bezeichnung	LokLnx	LokWin	EntLnxWin	EntWinLnx
Sender	Lenovo	Labor	Lenovo	Labor
Empfänger			Labor	Lenovo
OS	Linux	Windows	Linux/Windows	Windows /Linux
Protokoll	Mqtt, Mosquitto Broker auf MacMini, QoS 0			
Periode	500ms			
Test Wied.	3x		1x	
Netzwerk	LAN			
Weiteres	Synchronisation über Maschinenuhr		Nicht synchronisiert, Synchronisation über NTP	

Tabelle 7.1: Testparameter für Synchronisation mit NTP.

Lokale Tests

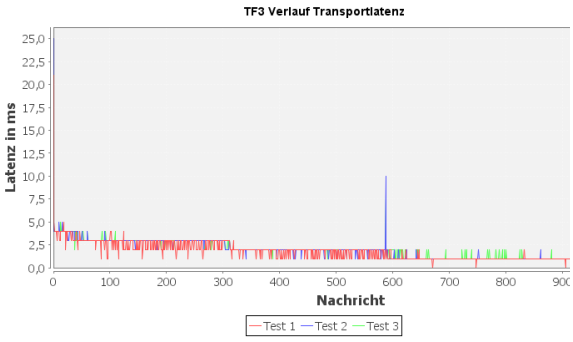


Tabelle 7.2: Latenzverlauf für LokLnx. Gemessene Latenz für jede Nachricht auf einem Ubuntusystem. Lokaler Testaufbau und drei Wiederholungen.

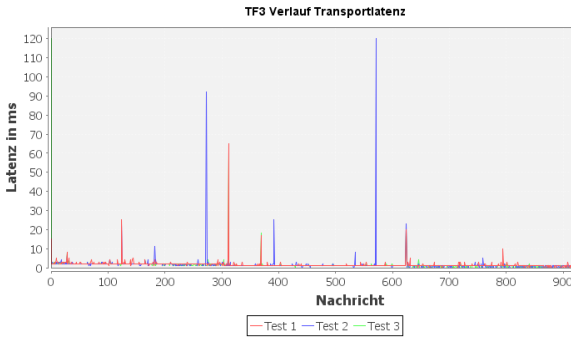


Tabelle 7.3: Latenzverlauf für LokWin. Gemessene Latenz für jede Nachricht auf einem Windowsystem. Lokaler Testaufbau und drei Wiederholungen.

Tabelle 7.17 und Tabelle 7.18 zeigen den Verlauf der Übertragunglatenz einer Nachricht für insgesamt 1000 Nachrichten und jeweils für ein Linux- (links) und ein Windowssystem (rechts). Die x-Achse repräsentiert die zeitlich geordneten Nachrichten. Nachricht 1 ist somit die erste und Nachricht 1000 somit die letzte gesendete Nachricht. Die y-Achse gibt die jeweilige Latenz in Millisekunden an. Da jeder Test auf einem System dreimal wiederholt wurde, gibt es drei Kurven in jeder Abbildung, eine je Durchgang. Alle Testdurchgänge für das Linuxsystem zeigen ähnliche Verläufe. Gleiches gilt für die verschiedenen Durchgänge auf dem Windowsrechner, wobei sich dort eine höhere Zahl von Ausreißern feststellen lässt. Diese tauchen jedoch an unterschiedlichen Stellen auf. Ferner sind die Amplituden dieser Scheitelpunkte mit maximal 120ms deutlich höher als auf dem Linuxsystem mit maximal 10 ms. Die Verläufe zeigen bei beiden Testrechnern eine lineare leicht abfallende Tendenz. Dabei gibt es jeweils Millisekundensprünge nach ungefähr der 300. und 600. Nachricht. Summa summarum kann man von identischen Verläufen sprechen.

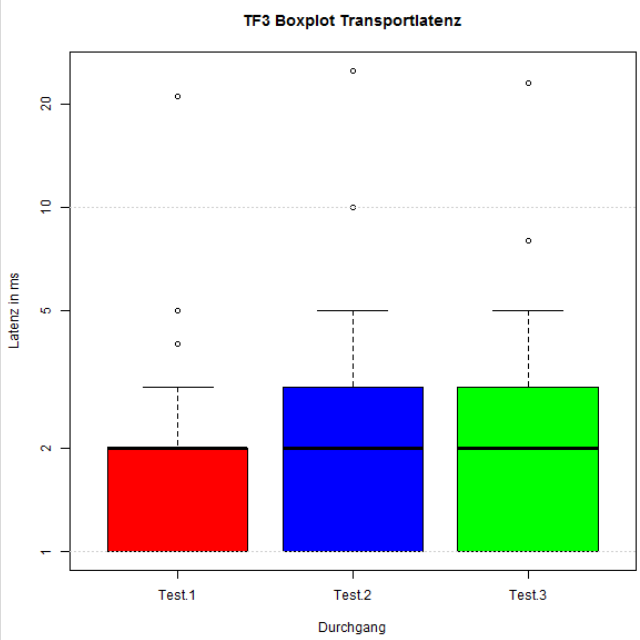


Tabelle 7.4: Box-Whisker-Plot für LokLnx. Verteilung kardinalskalierter Werte für drei Testdurchgänge auf einem Linuxrechner.

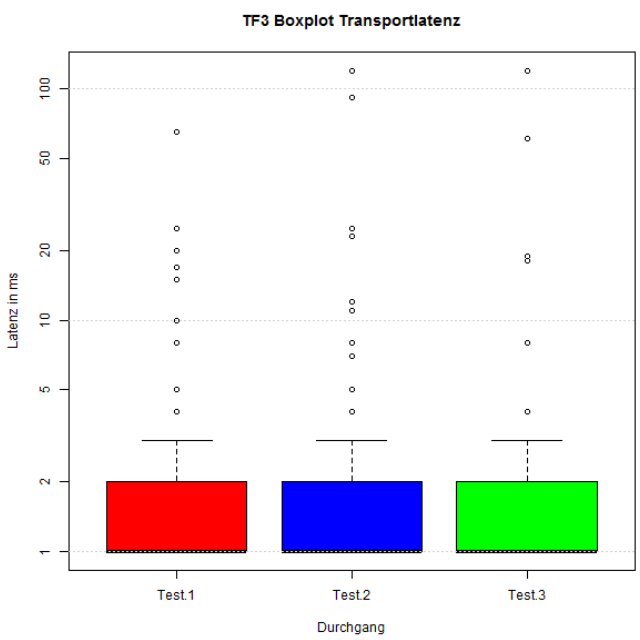


Tabelle 7.5: Box-Whisker-Plot für LokWin. Verteilung kardinalskalierter Werte für drei Testdurchgänge auf einem Windowsrechner.

Ein Blick auf Tabelle 7.4 und Tabelle 7.5 bestätigt die bisherige Einschätzung. Darin sind Kastendiagramme für die beiden lokalen Tests zu sehen, welche wie in Kapitel * beschrieben zu interpretieren sind. Für den Linuxrechner gilt: je nach Testdurchgang dauert in 50% der Fälle eine Übertragung ein bis zwei Millisekunden (Test1), beziehungsweise ein bis drei Millisekunden (Test2, Test3). Ausreißer gibt es ausschließlich nach oben. Die statistische Grenze für extreme Ausreißer liegt bei drei Millisekunden (Test1) und 5 Millisekunden (Test2, Test3). Die Anzahl jener ist gering und liegt unter einem Prozent. Die Testreihen auf dem Windowssystem ähneln sich im hohen Maße. Die Hälfte alle Latenzen beträgt ein bis zwei Millisekunden. Die Ausreißergrenze liegt in allen Testreihen bei 3 Millisekunden. Allerdings gibt es erkennbar mehr Ausreißer und wie bereits festgestellt, ist deren Maximalwert (über 100 Millisekunden) größer als auf dem Linuxrechner (unter 30 Millisekunden). Sie treten jedoch noch selten mit einer Wahrscheinlichkeit von einem Prozent und darunter auf.

Zusammenfassung lokale Tests

Untersucht wurde, in wie weit das eingangs beschrieben Szenario als geeignet gilt, um eine spätere Synchronisation getrennter Rechner und damit Systemuhren zu bewerten. Dafür wurde Testreihen auf den jeweiligen Rechner wiederholt durchgeführt. Je Rechner zeigen alle Testreihen konstante Ergebnisse, konkret, identische Latenzverläufe. Die jeweiligen Messwerte liegen zwischen ein und drei Millisekunden, die Ausreißergrenze knapp darüber bei maximal 5 Millisekunden. Zwar treten gelegentlich extreme Ausreißer auf und deren Häufigkeit, sowie deren Amplitude sind im bestimmten Maß abhängig vom Betriebssystem (auf Windowsrechner häufiger und mit höheren Werten, als auf Linuxrechner), jedoch erreicht die Wahrscheinlichkeit für einen extremen Ausreißer in allen Testreihen nie einen Wert über einen Prozent. Daher kann diese Beobachtung vernachlässigt und für weitere Testreihen können ähnlich konstante Ergebnisse erwartet werden. Das Szenario ist im hohen Maße determinierend und wird im Folgenden zur Bewertung der Synchronisation eingesetzt.

Entfernte Tests

Im Gegensatz zu den bisherigen Tests werden Sender und Empfänger nun auf unterschiedlichen Rechner platziert. Auf einem Rechner befinden sich ausschließlich Sender, auf dem anderen ausschließlich Empfänger. Für das Szenario wird nur je eine Instanz benötigt, sodass Nachrichten von genau einem Sender an genau einen Empfänger über einen Broker im Netzwerk verschickt werden.

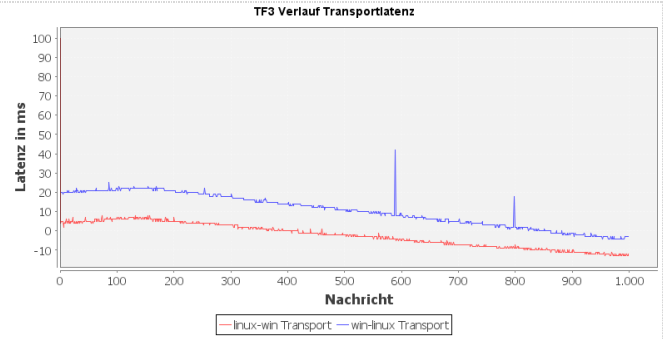


Tabelle 7.6: Verlauf der Übertragungslatenz für getrennte, nicht synchronisierte Rechner. Zwei dargestellte Kommunikationsrichtungen: Sender auf Linuxrechner, Empfänger auf Windowsrechner und umgekehrt.

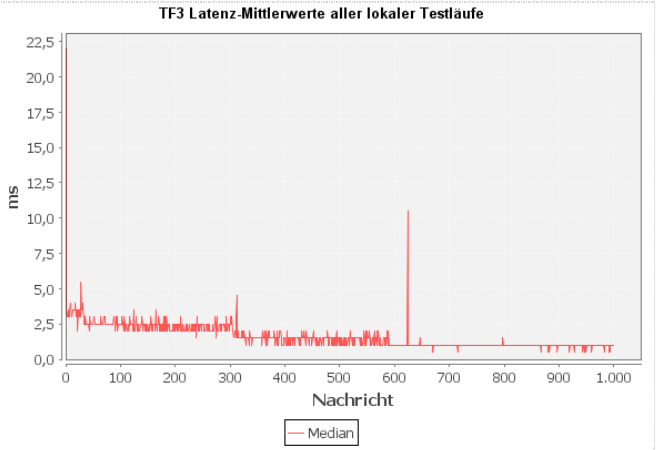
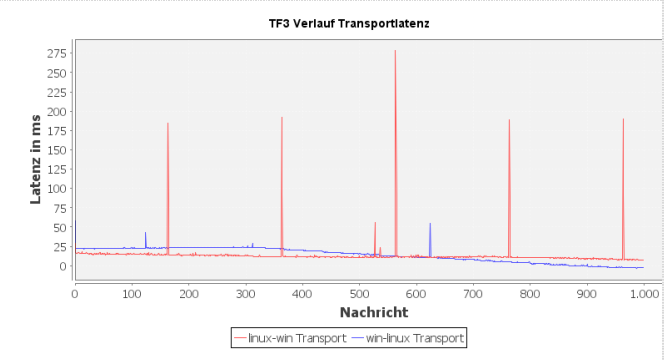


Tabelle 7.7: Erwarteter Verlauf der Übertragungslatenz. Mittelung aus allen lokalen Testreihen.

Tabelle 7.6 zeigt den Verlauf der Übertragungslatenzen für 1000 Nachrichten bei nicht synchronisierten Rechnern und für gegensätzliche Kommunikationsrichtungen. Tabelle 7.7 zeigt die erwartete Verlaufskurve und ist die Mittelung aus allen bisher durchgeführten Tests und dient als Vergleichsbasis. Beide Abbildungen zeigen deutliche Unterschiede. Man erkennt zum einen die deutlich abweichenden Latenzwerte, die von circa 25 Millisekunden bishin zu Werten unter null reichen, zum anderen dass es sich nicht mehr um eine linearen, nur leicht fallenden Anstieg handelt, sondern für beide Richtungen eine Schwingung mit Maximum zwischen der 100 und 200 Nachricht erkennbar ist, die nebenbei groß wesentlichere Anstiege zeigt. Die hohen und negativen Latenzwerte sind von Natur aus durch die unsynchronisierten Uhren bedingt, da ein zeitliches Ereignis auf dem beiden Rechnern einen unterschiedlichen Zeitstempel zugewiesen bekommt, der die Latenz bestimmt. Eine Ursache für die erkennbaren Schwingungen dürften die aktiven globalen Synchronisierungsmechanismen in beiden Rechnern sein. Beide Rechner synchronisieren sich nicht untereinander, jedoch mit Zeitservern im Internet. Dabei werden periodisch Phase und Frequenz der internen Zeitgeber angepasst, um eine Abdriftung der Uhren zu vermeiden. Dadurch entstehen die Anstiege in den Verlaufskurven.



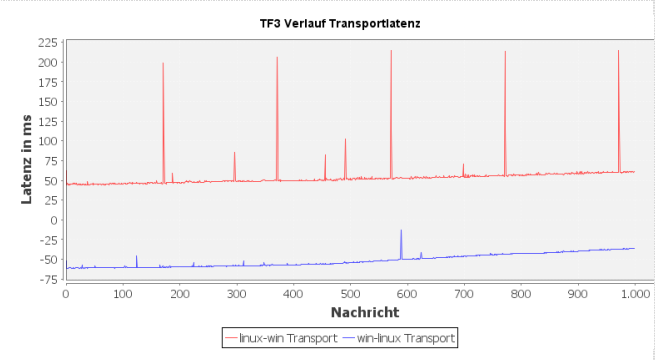


Tabelle 7.8: Text	Tabelle 7.9: Text

Die Rechneruhren werden nun wie in Kapitel * beschrieben synchronisiert. Ein Rechner, in diesem Fall der Windowsrechner fungiert als lokaler Zeitserver. Der Linuxrechner gleicht seine Uhr über eine LAN-Verbindung direkt mit diesem ab. Nach der Einrichtung können die Tests wiederholt werden. Tabelle 7.8 zeigt die Ergebnisse eines ersten Testdurchlaufs. Wiederum finden sich starke Abweichungen in den Latenzwerten, sowie negative Werte. Ferner sind Schwingungen erkennbar, insbesondere bei Nachrichtenübertragung vom Windowsrechner hin zum Linuxrechner. Da NTP eine Zeit lang benötigt Uhren zu synchronisieren, wird der Test zu einem späteren Zeitpunkt wiederholt unter der Annahme. Dessen Ergebnisse sind in Tabelle 7.9 zu sehen. Diese zeigen keine Verbesserung hinsichtlich der zu erwartenden Messwerte. Im Gegenteil, die Spannweite der Latenzen ist deutlich höher als zuvor und reicht von weniger als -50 Millisekunden bis mehr als 50 Millisekunden. Wiederum sind Anstiege in den Verläufen sichtbar.

Die gesammelten Ergebnisse lassen erkennen, dass die vorgestellte Methode aus Kapitel * nicht ausreichend gut funktioniert, um die Rechner hinsichtlich der Testanforderungen zeitlich zu synchronisieren. Ferner wurde versucht, durch Anpassung zur Verfügung stehender Parameter bessere Ergebnisse zu erzielen. So wurde die Aktualisierung- und Abgleichsfrequenz zwischen den Rechnern erhöht. Diese Maßnahmen konnten keine signifikante Veränderung bewirken.

Da einige Quellen jedoch die Tauglichkeit von NTP als Synchronisationsmechanismus im LAN bestätigen *, erfolgte eine weitere Testreihe mit einer alternativen NTP-Implementation für Windows.

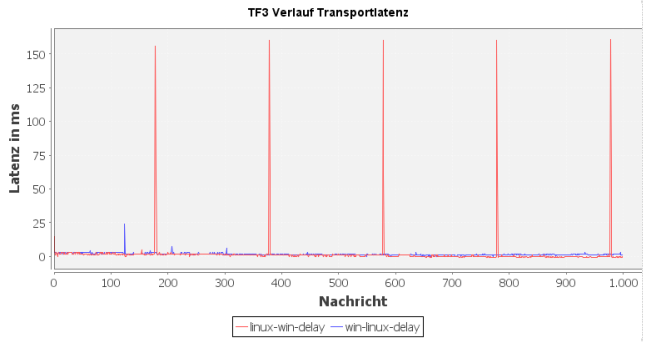


Tabelle 7.10: Text

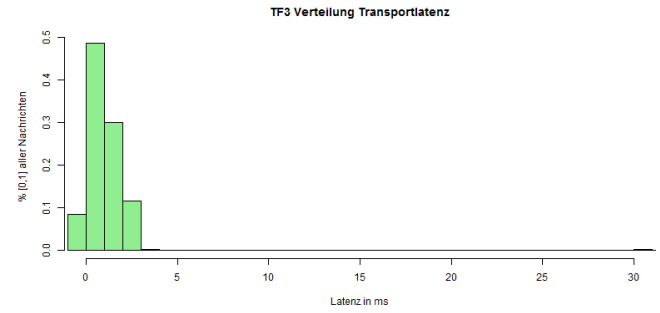


Tabelle 7.11: Text

Tabelle 7.10 visualisiert die Entwicklung der Übertragungslatenzen mit der *Meinberg* NTP-Implementation. Bereits der erste Durchlauf vermittelt ein ähnliches Bild wie Tabelle 7.7 und damit die zu erwartenden Testergebnisse. In Tabelle 7.11 ist ein Histogramm zu erkennen über die Verteilung der Messwerte in Intervallen von einer Millisekunde. Darin sieht man, dass nach wie vor negative Werte erfasst wurden. Immerhin fast jede 10 Latenz ist kleiner als null.

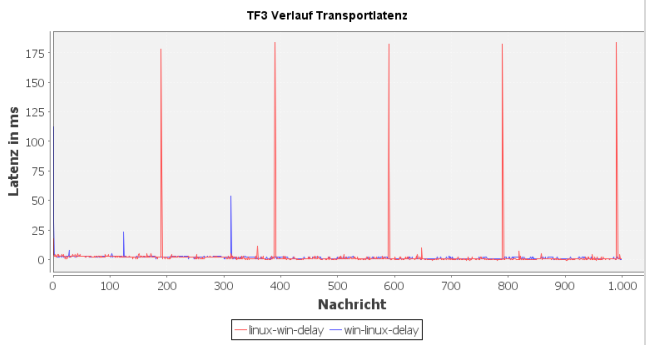


Tabelle 7.12: Text

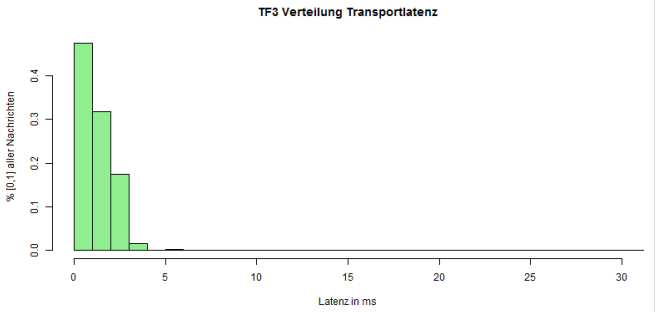


Tabelle 7.13: Text

Aufgrund der fortlaufenden Uhrensynchronisation und der zu erwartenden Verbesserung wird der Test zu einem späteren Zeitpunkt wiederholt. Die Ergebnisse sind in Tabelle 7.12 und Tabelle 7.13 zu sehen. Zum einen zeigt sich, dass beide Testreihen bisher recht genau den erwartenden Ergebnissen entsprechen. Zum anderen tauchen in letzterer keine negativen Ergebnisse auf. Die Latenzverläufe werden, wie bei den lokalen Tests, durch lineare Geraden beschrieben ohne nennenswerte Anstiege. Ebenso gleichen sich statistische Werte, wie in Tabelle 7.14 zu erkennen. 50% aller Werte sind kleiner gleich 2 Millisekunden. Die Grenze zwischen schwachen und extremen Ausreißern befindet sich bei 3 Millisekunden. Die Häufigkeit extremer Werte liegt bei 1% und darunter.

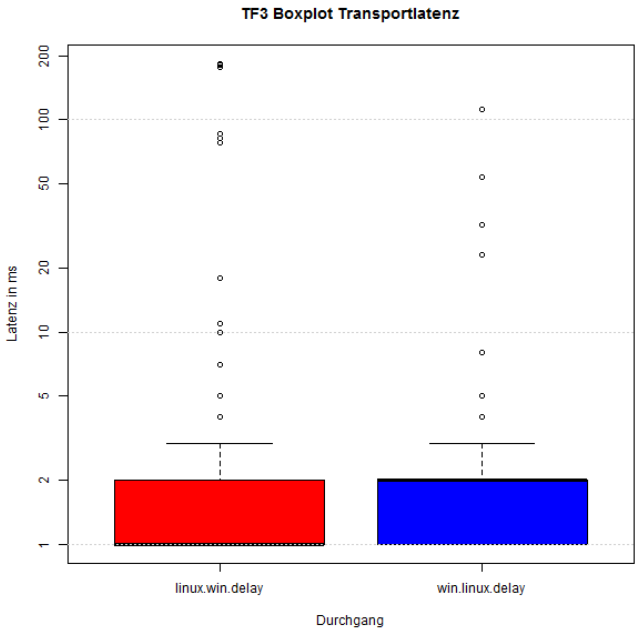


Tabelle 7.14: Text

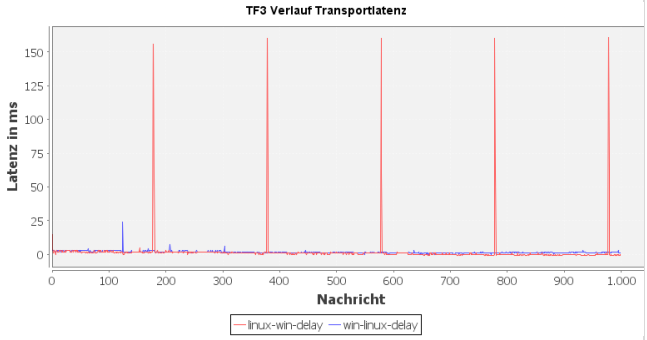


Tabelle 7.15: Text

Zusammenfassung entfernte Tests

Nachdem die Eignung des vorgestellten Szenarios festgestellt wurde, konnte daraufhin die Eignung von NTP als Synchronisationsmechanismus untersucht werden. Die Ergebnisse zeigen, NTP ist für die Synchronisation im LAN geeignet, jedoch eingeschränkt. Zum einen ist es von Bedeutung, welche Implementation schlussendlich genutzt wird. Win32tm, als Standarddienst auf Windowssystem konnte keine genügend exakte Synchronisation ermöglichen, die alternativen Meinberg-Implementation dagegen schon. Für Linux, respektive Ubuntu ist die zur

Verfügung gestellte NTP-Bibliothek ausreichend. Zum anderen ist der Grad der Synchronisation Schwankungen unterworfen, da die verschiedenen Systemuhren voneinander abdriften können und die Anpassung kontinuierlich und in bestimmten Intervallen erfolgt. Es empfiehlt sich NTP eine Zeit lang agieren zu lassen. Dadurch ist ein höheres Maß an Synchronisation zu erwarten. Tabelle 7.15 zeigt den gemessenen Versatz der Rechneruhren über einen bestimmten Zeitraum. Die Abszissenachse ist nicht zeitlich skaliert. Die Ordinatenachse zeigt den Versatz der Uhren in Millisekunden. NTP ist nicht über den gesamten Darstellungsverlauf aktiv. Vor den Anfragen 1, 310, 360 und 420 wurde NTP eine Zeit lang deaktiviert. Man erkennt, dass bei der darauffolgenden Anfrage die Uhren etwas auseinander gedriftet sind, NTP sich jedoch relativ schnell in einem Bereich unter einer Millisekunde Abweichung einpendelt. Die Synchronisation ist folglich mit einer Auflösung unter einer Millisekunde realisierbar.

Zwei Besonderheiten sind abseits der Synchronisation fest zu stellen. Erstens zeigt das lokale Szenario auf Windowsrechner häufiger extreme Ausreißer mit ebenso stärkeren Amplituden. Zweitens zeigen sich bei der Nachrichtenübertragung von Linuxrechnern hin zu Windowsrechner periodisch auftretende und ähnlich hohe Spitzenwerte (Tabelle 7.8, Tabelle 7.9, Tabelle 7.10, Tabelle 7.12). Eine Ursache können Anwendungen im Hintergrund sein, die aus welchen Gründen auch immer die Übertragung verzögern. Tatsächlich konnten nach einem Austausch der Testrechner – Dell anstatt Windows – und wiederholten Test keine periodischen Ausreißer mehr festgestellt werden. Die Ursache lag demnach innerhalb des Systems des vorherigen Rechners.

7.1.3 *AUSWERTUNG*

7.2 VERBINDUNGSAUFBAU

Im Folgenden richtet sich der Fokus auf die Dauer, die ein Teilnehmer benötigt, um eine Nachricht zu versenden beziehungsweise zu empfangen oder abzufragen. Dabei interessiert nicht die Übertragung der Nachricht allein, sondern der Vorgang, der Anmelden, Registrierung, Absenden, Empfangen zusammenfasst. Die einzelnen Teilschritte sind je nach Protokoll unterschiedlich komplex und nicht überall notwendig. Daher sind abweichende Zeiten zu erwarten.

Die Motivation für diese Testreihe ergibt sich aus den der Betrachtungen der Szenarien. Ganz im Sinne von *IoT* erwartet man eine zunehmende Vernetzung von einer Vielzahl kleinen kommunikationsfähigen Rechnern. Diese besitzen nicht selten eine mobile Energiequelle und müssen daher energieeffizient arbeiten. Oftmals geht es um das Sammeln von Daten und einer periodischen Übertragung dieser. Die Annahme geht davon aus, dass über weite Strecken keine Kommunikation stattfindet. Allein deswegen kann es Sinn ergeben, eine Verbindung zwischen Kommunikationsperioden zu beenden und damit Energie zu sparen. Ferner werden dadurch Ressourcen im Netzwerk wieder freigegeben.

Auf der anderen Seite bedeutet dies, dass vor jeder Kommunikation eine logische Verbindung sichergestellt sein muss. Ja nach Protokoll und OSI-Schicht unterscheidet sich der Aufwand dafür. Eine Ausnahme bilden verbindunglose Protokolle in paketvermittelnden Netzwerken, zum Beispiel UDP über Ethernet. Getestet werden *Mqtt*, *Xmpp* und *CoAP*.

7.2.1 VORBETRACHTUNG

Gemessen wird die Zeit vom Verbindungsaufbau ab bis zum Empfang der Nachricht. Streng genommen wird zwischen der Verbindungsaufbaulatenz und der Transportlatenz unterschieden. Es gibt bei allen drei Protokollen dahingehend wenig Gemeinsamkeiten. Mqtt und XMPP nutzen für den Transport TCP. Vor der eigentlichen Datenübertragung muss eine Sitzung aufgebaut werden. CoAP nutzt UDP und daher ist eine umgehende Datenübertragung möglich.

Auf Anwendungsebene erzeugen alle drei Protokolle eine Sitzung. Bei CoAP wird dabei lediglich ein logischer Datagrammkanal zum Server erzeugt. Dabei wird vorranging die Erreichbarkeit geprüft und Ressourcen für die folgende Verarbeitung erzeugt. Mqtt erzeugt neben TCP-Verbindung eine logische Sitzung, bei der neben dem Namen des Teilnehmers ebenso Sitzungsparameter übertragen werden. Optional kann eine Authentifikation notwendig sein. XMPP verhält sich ähnlich, wobei eine Authentifikation die Regel ist. Ferner werden alle Daten nicht binär, wie bei Mqtt und CoAP, sondern als Text und in XML-Befehlen übertragen, wobei deutlich mehr Zeichen übertragen werden müssen. Die Aushandlung einer Sitzung ist deutlich komplexer.

Aus den Ausssagen ergibt sich die Erwartung, dass für CoAP auf Grund des verbindungslosen Transports und der einfachen Sitzungserzeugung die geringsten Latenzen zu erwarten sind. Danach folg Mqtt mit dem komplexeren TCP. XMPP dürfe zumindest bei dem Aufbau eine Verbindung deutliche Zeiteinbußen aufzeigen. Für den Transport nutzt es ebenso TCP. Ergänzend wird für Mqtt und CoAP untersucht, wie sich die Transportlatenz bei zuverlässiger Übertragung verhält. Die Zuverlässigkeit sollte keinen zeitlichen Einfluss auf die Erzeugung einer Verbindung haben.

7.2.2 ABLAUF

Alle Tests finden auf demselben Rechner und im selben Netzwerk statt. Unabhängig vom Protokoll werden jeweils 1000 Clients erzeugt. Nacheinander meldet sich ein Client beim entsprechenden Server an, registriert sich gegeben falls für ein Ressource, sendet eine Nachricht an diese Ressource und wartet auf die Auslieferung dieser Nachricht. Danach meldet sich der Client wieder ab. Nach einer bestimmten Zeitspanne meldet sich der nächste Client an und wiederholt die Prozedur. Es werden jeweils die Zeitpunkte vor und nach der Anmeldung – die Verbindungsaufbaulatent - und die Zeitpunkte vor Sendern und nach Erhalt der Nachricht – die Übertragungslatzenz registriert. Eine Ressource ist je nach Protokoll ein CoAP-Ressource, ein Mqtt-Topic oder ein Xmpp-Multiuser-Chat.

- Abbildung Testablauf

7.2.3 AUFBAU

Tabelle 7.16 fasst die unterschiedlichen Parameter für alle Test zusammen. Insgesamt werden sechs Testreihen durchgeführt. Drei Mal kommt Mqtt mit den möglichen QoS-Parametern zum Einsatz. Xmpp wird in Kombination mit einer unzuverlässigen Multiuser-Chat Variante implementiert und hinsichtlich CoAP betrachten wir eine zuverlässige und eine nicht zuverlässige Übertragung. Die Clients befinden sich in jedem Testdurchgang auf dem Lenovo-Notebook. Ein Client ist Sender und Empfänger in einem. Auf der Serverseite kommen OpenFire für Xmpp, Mosquitto für Mqtt und jCoap als CoAP-Server zum Einsatz. Die

Instanzen kommunizieren über das lokale Netzwerk. Jede Testreihe wird drei Mal wiederholt.

Bezeichnung	Mqtt_0	Mqtt_1	Mqtt_2	Xmpp	CoAP	CoAP_r
Sender	Lenovo					
Empfänger						
OS	Linux/Ubuntu					
Protokoll	Mqtt			Xmpp	CoAP	
Server	MacMini, Moqsuitto			MacMini, OpenFire	MacMini, jCoap Server	
Periode	500ms					
Test Wied.	3x					
Netzwerk	LAN					
Weiteres	QoS 0 Best effort	QoS 1 Most once	QoS 2 Exactly once	unzuverlässig	unzuverlässig	zuverlässig

Tabelle 7.16: Testparameter für Verbindungaufbau mit unterschiedlichen Protokollen und Übertragungsparametern.

7.2.4 Ergebnisse

Verbindungsaufbau mit Mqtt

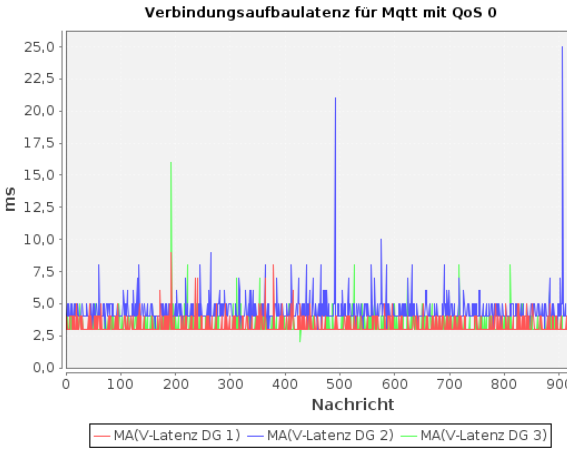


Tabelle 7.17: Verlauf der Verbindungsaufbaulatenz für Mqtt_0. Gemessene Zeitspanne zwischen Start und Bestätigung der Anmeldungen. 1000 Clients und drei Durchläufe.

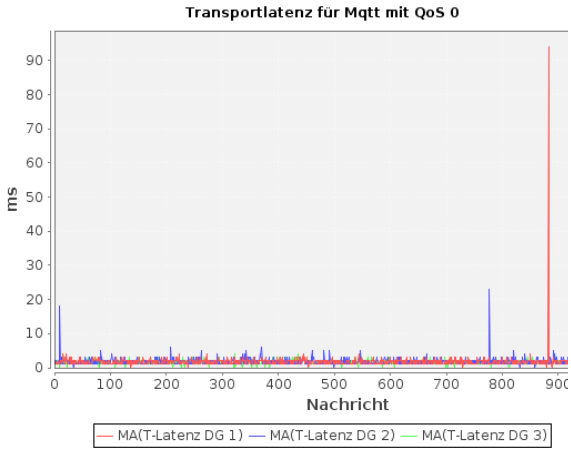


Tabelle 7.18: Verlauf der Übertragungslatenz für Mqtt_0. Gemessene Zeitspanne zwischen Versand und Erhalt einer Nachricht. 1000 Clients und drei Durchläufe.

Tabelle 7.17 und Tabelle 7.18 zeigen jeweils die Verläufe für die Verbindungsaufbaulatenz und Übertragungslatenz für den Testfall Mqtt_0. Die drei dargestellten Verläufe den Abbildungen repräsentieren je einen Testdurchlauf. In Jedem Testdurchlauf versendet jeder der 1000 Clients zeitlich voneinander getrennt eine Nachricht. Die Abszisse repräsentiert die jeweilige Nachricht und die Ordinate gibt die dazugehörige gemessene Latenz an.

Für zwei von drei Testdurchläufen pendelt die gemessene Dauer für den Aufbau einer Verbindung zwischen 3 und 4 Millisekunden, bei der dritten zwischen 4 und 5 Millisekunden. Die Werte sind dabei über den gesamten Zeitraum konstant. Nur vereinzelt weichen Messwerte ab, jedoch unregelmäßig und in jeder Testreihe an unterschiedlichen Stellen. Für die Dauer der Nachrichtenübertragung gilt ähnliches. Für alle drei Testreihen schwankt die Übertragungsdauer zwischen 1

und 3 Millisekunden. Ausreißer gibt es selten. Die Messkurven fallen weder noch steigen sie.

Im Folgenden überspringen wir die Betrachtungen der einzelnen Testreihen je Tests, da weder für Mqtt, Xmpp noch CoAP im Vergleich der Testreihen zueinander keine Auffälligkeiten auftraten. Stattdessen werden die Testreihen für jeden Test gemittelt. Im Anhang * befinden sich die ausgelassenen Diagramme.

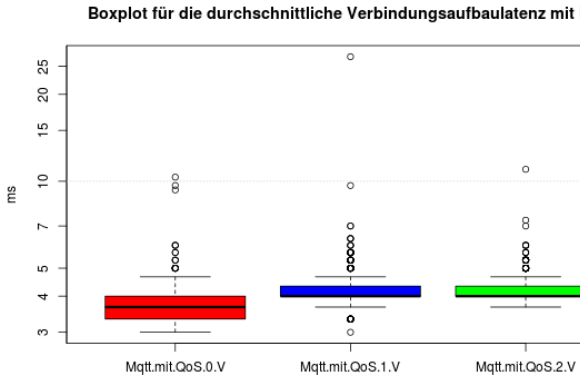


Tabelle 7.19: Boxplot der Latenz zum Aufbau einer Verbindung mit Mqtt und unterschiedlichen QoS-Parametern

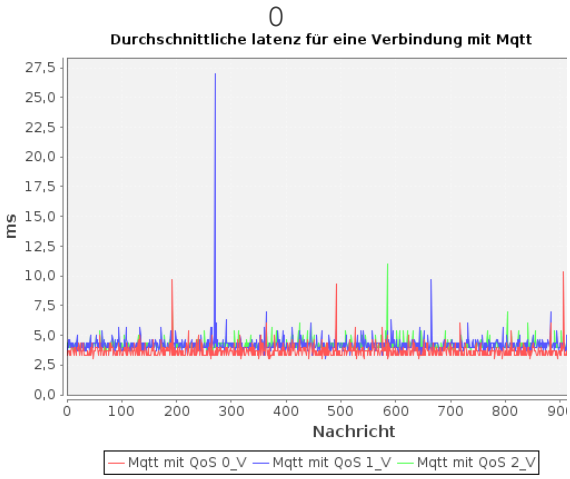


Tabelle 7.20: Verlauf der Verbindungsaufbaulatenz für Mqtt mit unterschiedlichen QoS-Parametern. Gemessene Zeitspanne zwischen Aufbau und Bestätigung.

Tabelle 7.19 zeigt drei Boxplots mit den durchschnittlichen Latenzen zum Aufbau einer Verbindung mit Mqtt und unterschiedlichen Qualitätsparametern. Mit Mqtt besteht die Möglichkeit einer ungesicherte (QoS.0), eine garantierte (QoS.1) und eine garantierte und dopplungsfreie (QoS.2) Auslieferung zu gewährleisten. Die Auslieferungsqualität wirkt sich nicht signifikant auf die Dauer zum Aufbau einer Verbindung aus. Die Mehrheit der Messwerte liegt zwischen 3 und 5 Millisekunden für den Aufbau einer Verbindung. Allerdings zeigt sich für die garantierte Auslieferung eine leichte Verschiebung der Messwerte nach oben. Ausreißer gibt es in allen drei Testreihen, jedoch selten und mit einer Wahrscheinlichkeit unter einem Prozent.

Tabelle 7.20 zeigt den Verlauf der durchschnittlichen Latenzen zum Aufbau einer Verbindung mit Mqtt. Die drei dargestellten Kurven repräsentieren wiederum die drei Qualitätsparameter für die Auslieferung. Die Darstellung bestätigt die Aussagen aus dem vorrangegangenen Abschnitt und zeigt auch im Verlauf der Kurven keine Auffälligkeiten. Diese bewegen sich konstant in einem Bereich von 3 bis 5 Millisekunden ohne deutliche Anstiege oder Abfälle. Ausreißer treten spontan ohne erkennbare Regelmäßigkeit auf.

Nachrichtenübertragung mit Mqtt

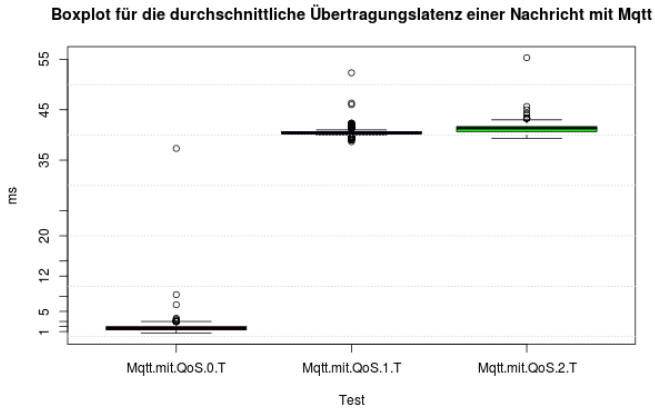


Tabelle 7.21: Boxplot der Latenz beim Übertragen einer Nachricht mit Mqtt und unterschiedlichen QoS-Parametern

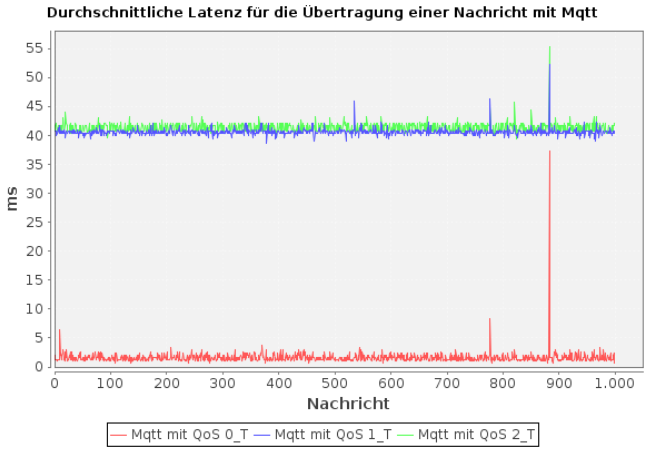


Tabelle 7.22: Verlauf der Übertragungslatenz für Mqtt mit unterschiedlichen QoS-Parametern.

In Tabelle 7.21 ist nun zu sehen, wie sich die verschiedenen Qualitätsparameter auf die Transportlatenz auswirken. Nachrichten über eine unsichere Verbindung werden ohne große Verzögerungen übertragen. Die gemessene Laufzeit beträgt in der Regel 1 bis 3 Millisekunden. Die Messwerte ändern sich deutlich, wenn eine zuverlässige Verbindung genutzt wird. Es spielt dabei weniger eine Rolle, ob eine Nachricht *mindestens* oder *genau ein Mal* übertragen wird. Bei beiden Varianten steigt die Laufzeit auf 40 bis 43 Millisekunden. Ausreißer gibt es in allen drei Testreihen wenige und wenn, dann fast ausschließlich nach oben.

Tabelle 7.22 zeigt die Verläufe der gemessenen Latenzen über die Zeit hinweg. Wieder zeigen sich konstant bleibende Werte ohne Anstiege oder Absenkungen. Der Großteil der Werte schwankt im Millisekundenbereich um den jeweiligen Mittelwert. Wie in der Abbildung zuvor die deutliche höhere Übertragungslatenzen bei einer zuverlässigen Verbindung. Auffällig sind die Ausschläge um die 890 Nachricht herum. Alle drei Kurven lassen einen deutlichen Ausreißer erkennen.

Verbindungsaufbau mit Xmpp

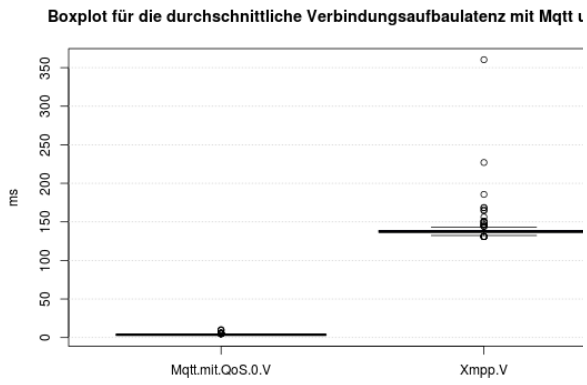


Tabelle 7.23: Boxplot der Latenz beim Aufbau einer Verbindung mit Mqtt und Xmpp im Multi-User Chat.

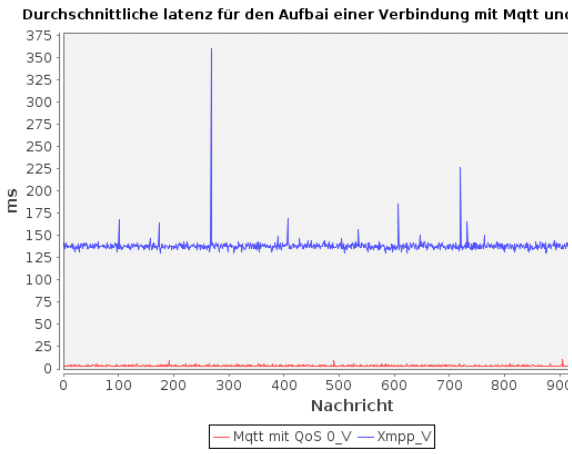


Tabelle 7.24: Verlauf der Latenz beim Aufbau einer Verbindung mit Mqtt und Xmpp im Multi-User Chat für 1000 Clients.

Nach *Mqtt* wird nun *Xmpp* als Kommunikationsprotokoll eingesetzt. Die Implementation als Multi-User Chat überträgt die Nachrichten standardmäßig

unzuverlässig. Daher empfiehlt sich für einen Vergleich die Ergebnisse aus der unzuverlässigen Übertragung mit Mqtt heranzuziehen (Mqtt_0). In Tabelle 7.23 werden die Boxplots für die durchschnittliche Dauer für den Aufbau einer Verbindung mit Mqtt und Xmpp dargestellt. Es ist erkennbar, dass Mqtt in deutlich kürzerer Zeit eine Verbindung herstellt. Mit Medianwerten von 3,3 zu 137,6 Millisekunden dauert der Prozess mit Xmpp rund 45 Mal so lang, wie mit Mqtt. Quartil und Whisker-Antenne sind sichtbar weiter entfernt vom Medianwert als bei Mqtt. Daher kann von einer größeren Streuung der Messwerte ausgegangen werden. Xmpp zeigt eine ebenso größere Zahl von Ausreißer, welche erkennbar vom Median abweichen.

Der Eindruck wird von Tabelle 7.24 bestätigt. Beide Latenzkurven verhalten sich relativ konstant über die Zeit hinweg. Es gibt keine sichtbaren Anstiege oder Gefälle. Jedoch bewegt sich die Xmpp-Kurve in größeren Schwingungen um den Mittelwert herum. Ferner sind die zahlreicheren Ausreißer an den Spitzen zu erkennen. Diese treten unregelmäßig auf.

Nachrichtenübertragung mit Xmpp

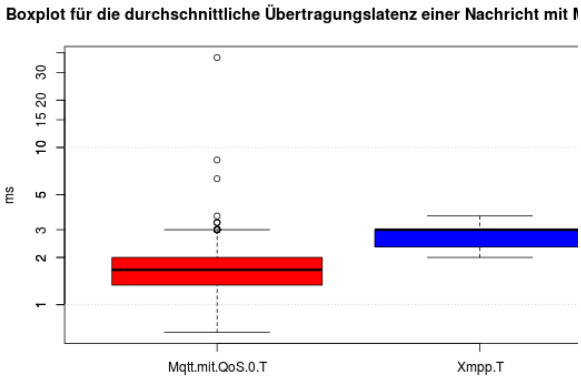


Tabelle 7.25: Boxplot der Latenz beim Übertragen einer Nachricht mit Mqtt und Xmpp im Multi-User Chat.

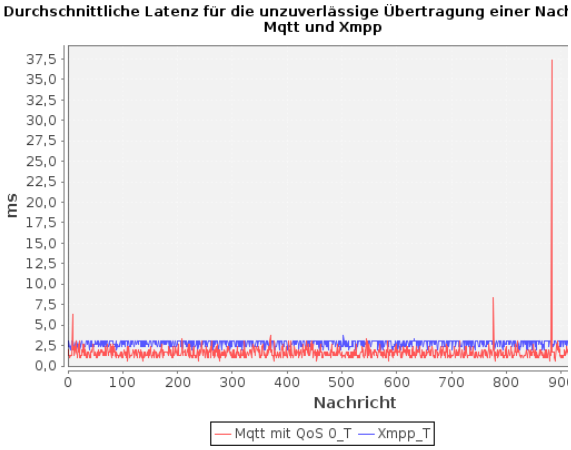


Tabelle 7.26: Verlauf der Latenz beim Übertragen einer Nachricht mit Mqtt mit Xmpp im Multi-User Chat für 1000 Clients.

Für die Latenz bei der Übertragung einer Nachricht ergibt sich ein anderes Bild. Hier verhält sich Xmpp annähernd gleich zu Mqtt. Die Mehrzahl der Messwerte lag in den Ergebnissen zwischen fast 0 bis 5 Millisekunden, im Kernbereich zwischen 1,5 und 3 Millisekunden, wie Tabelle 7.25 zeigt. Insgesamt benötigt Xmpp im Schnitt 1 Millisekunde länger um Nachrichten zu übertragen. Ausreißer gibt es diesmal auf der Seite von Mqtt, wober diese zahle- und wertemäßig nicht ins Gewicht fallen.

Die Verlaufskurven beider Protokolle in Tabelle 7.26 verhalten sich konstant über die Zeit. Aus den wenigen erkennbaren Ausreißern lassen sich keine Unregelmäßigkeiten schließen.

Verbindungsaufbau mit CoAP

Tabelle 7.27: Boxplot der Latenz Aufbau einer Verbindung mit CoAP.

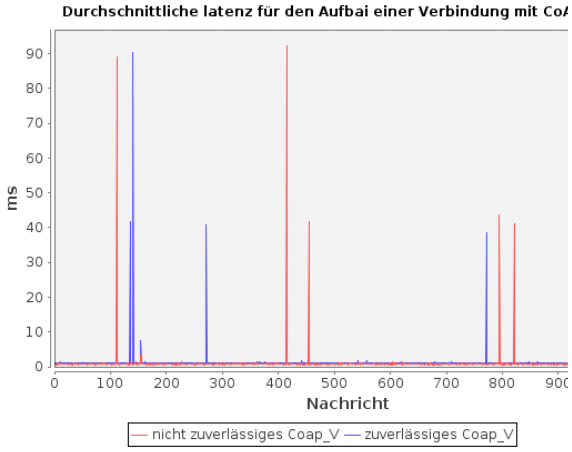


Tabelle 7.28: Verlauf der Latenz beim Aufbau einer Verbindung mit CoAP und 1000 Clients.

In Tabelle 7.27 ist zu sehen, wie wieviel Zeit im Schnitt ein Verbindungsaufbau mit CoAP in Anspruch nimmt. Dabei wird zwischen einer zuverlässigen und einer nicht zuverlässigen Übertragung unterschieden. Die dargestellten Boxplots sind annähernd identisch. Auffallend ist, dass es keine erkennbaren Quartil- und Whiskerstufen gibt. Damit sind fast alle Messwerte deckungsgleich mit dem jeweiligen Medianwert. Der Verbindungsaufbau mit zuverlässiger Übertragung dauert nur einen Bruchteil länger. In der Summe liegen bei beiden CoAP-Konfigurationen die Latenzen bei einer Millisekunden und darunter. Zwar gibt es wenige Ausreißer, jedoch können diese fast das 100-fache des Medianwertes annehmen.

Tabelle 7.28 zeigt den Verlauf der Latenzkurve beim sequentiellen Aufbau einer Verbindung von 1000 Clients. Die Kurven verhalten sich bei der zuverlässigen, wie bei der nicht zuverlässigen Verbindung relativ konstant und werden unregelmäßig von deutlichen Ausreißerwerten begleitet. Anstiege oder Gefälle sind nicht zu erkennen.

Nachrichtenübertragung mit CoAP

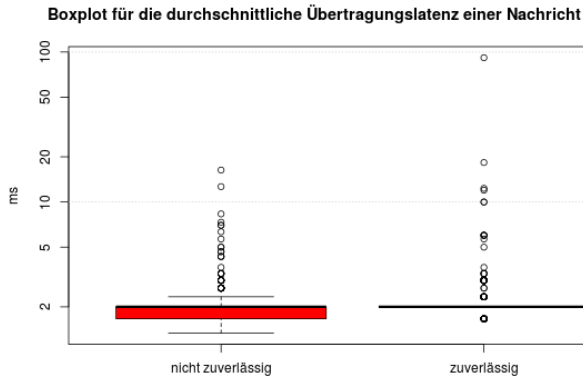


Tabelle 7.29: Boxplot der Latenz beim Übertragen einer Nachricht mit CoAP.

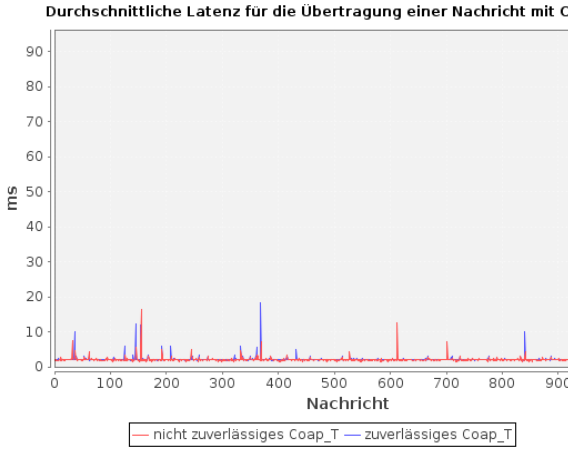


Tabelle 7.30: Verlauf der Latenz beim Übertragen einer Nachricht mit CoAP und 1000 Clients.

Bleibt die Betrachtung der Transportlatenz. Die Boxplots für eine zuverlässige und unzuverlässige Übertragung mit CoAP sind annähernd identisch wie Tabelle 7.29 zeigt. Der Großteil der gemessen Werte seigt nicht über 2 Millisekunden hinaus. In

beiden Messreihen gibt es ein ähnliches Aufkommen an Ausreißer, das bei einer Wahrscheinlichkeit von annähernd einem Prozent liegt.

Die Verlaufskurve beider Testreihen in Tabelle 7.30 zeigt konstante Verläufe ohne erkennbare Anstiege oder Absenkungen. Ausreißer treten unregelmäßig auf. Die Messwerte bewegen sich um den Mittelwert herum.

7.2.5 *ZUSAMMENFASSUNG*

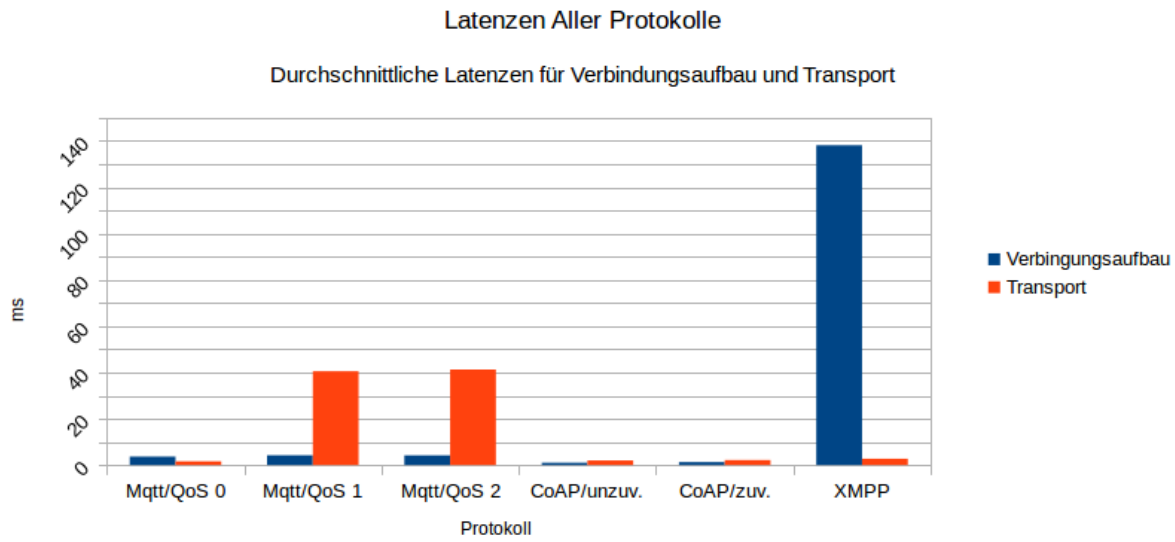


Tabelle 7.31: Überblick über die gemittelten Latenzen für die Aufbau einer Verbindung und der Übertragung einer Nachricht. Getestet wurden Mqtt mit den QoS-Parameter 0,1 und 2, Xmpp mit einem unzuverlässigen Multi-User Chat und CoAP mit zuverlässiger und nicht zuverlässiger Übertragung.

Verbindungsaufbau

Vergleicht man nun im Folgenden die Messwerte aller Testvarianten ergibt sich Tabelle 7.30. Im Bild sind die gemittelten Latenzen für den Aufbau einer Verbindung dargestellt (blau). Die Mittelwerte lassen sich den Protokollen nach sortieren. CoAP benötigt die geringsten Zeitspannen, um eine Verbindung zum Server aufzubauen. Knapp dahinter folgt Mqtt. Weit abgeschlagen dagegen ist Xmpp. Sind die Mittelwerte bei den anderen Protokollen stets unter 5 Millisekunden dauert ein Verbindungsaubau mit Xmpp fast 140 Millisekunden und damit rund 30 Mal so lange. Ferner gibt es keinen erkennbaren Einfluss der Qualitätsparameter eines Protokolls auf die Verbindungslatenz. Weder CoAP, noch Mqtt zeigen eine Zunahme der Latenz bei einer gesicherten Übertragung. Ein Blick auf die Statistische Auswertung und Tabelle 7.32 bestätigt den Eindruck. Darin sind der minimale, der maximale Wert, der Mittelwert, die Standardabweichung, der Median und Anzahl der verlorenen Nachrichten aufgelistet. Xmpp weist bei allen Kategorien die schlechtesten Werte auf und das mit Abstand. Zwar sind bei der Übertragung keine Nachrichten verloren gegangen. Dies gilt jedoch für alle Tests. Mittel- und Mediawert sind bei allen Protokollen sehr niedrig. CoAP hat deutlich hähere Maxima und Standardabweichungen als Mqtt.

Test	Mqtt_0	Mqtt_1	Mqtt_2	Xmpp	CoAP	CoAP_r
Minimum	2,7	3	3	129,7	0,3	0
Maximum	16,7	30,7	14,3	451	271	215,7
Mittelwert	3,4	4,2	4,2	138,2	1	1,3
Standard-abweichung	0,8	1,2	0,7	11	9,4	7,3
Median	3,3	4,3	4	137,7	0,7	1
Verlorene Nachrichten	0	0	0	0	0	0

Tabelle 7.32: Statistische Werte in Millisekunden für die Dauer bis zum Aufbau einer Verbindung.

Nachrichtenübertragung

Anders verhält es sich mit der Latenz für die Übertragung einer Nachricht. Hier kann keine Staffelung nach Protokoll festgestellt werden. CoAP, Xmpp und unzuverlässiges Mqtt übertragen Nachrichten mit fast identischen Transportlatenzen von 3 bis 4 Millisekunden. Wird eine zuverlässige Mqtt-Verbindung aufgebaut erhöht sich die Latenz auf rund 40 Millisekunden und damit fast das 10-fache. Dabei spielt es keine Rolle, ob Nachrichten mindestens (QoS 1) oder exakt ein Mal (QoS 2) übertragen werden. Das heißt: Xmpp verhält sich bei der Übertragung so gut wie CoAP und unzuverlässiges Mqtt. Zuverlässigkeit wirkt sich nur auf Mqtt und die dort gemessenen Übertragungslatenzen aus. Ein Blick auf die statistischen Werte in Tabelle 7.33 zeigt ein relativ ausgeglichenes Bild. Das vorherige schlechte Abschneiden von Xmpp ist darin nicht zu erkennen. Im Gegenteil, Xmpp zeigt konstant stabile Ergebnisse mit einem niedrigen Maximum und niedriger Standardabweichung. Negativ hervor sticht Mqtt bei einer zuverlässigen Übertragung.

Test	Mqtt_0	Mqtt_1	Mqtt_2	Xmpp	CoAP	CoAP_r
Minimum	0	38	39	2	1	1
Maximum	44,7	58,3	57,7	4,3	27	103
Mittelwert	1,6	40,6	41,3	2,7	2	2,2
Standard-abweichung	1,7	0,9	1,1	0,4	1,1	3,4
Median	1,3	40,3	41,7	3	2	2
Verlorene Nachrichten	0	0	0	0	0	0

Tabelle 7.33: Statistische Werte in Millisekunden für die Dauer einer Nachrichtenübertraung.

Bewertung

Die oben genannten Erkenntnisse müssen im Kontext des gewählten Szenarios bewertet werden und sind nicht als allgemein gültige Aussagen zu verstehen. Die Testbedingungen waren fast ideal und daher gelten die Aussagen lediglich für ideale Testbedingungen. Dazu zählen ein lokales zuverlässiges Hochgeschwindigkeitsnetzwerk, kleine Datenmengen, so gut wie kein Netzwerkverkehr und damit keine Netzwerklast. Ferner können Schwankungen im Millisekundenbereich ihre Ursache in der Auflösung der Zeitgeber haben, die im konkreten Fall 1 Millisekunde beträgt. Umso schwerer wiegen auf der anderen Seite signifikante Unterschiede in den Messergebnissen bei idealen Verbindungseigenschaften. Dazu zählen insbesondere die 30-fach längeren

Verbindungszeiten von Xmpp zu allen anderen Protkollen und die 10-fach längeren Übertragungszeiten von Mqtt bei einer zuverlässigen Verbindung.

7.3 ACDSENSE-INTEGRATION

7.3.1 VORBETRACHTUNG

ZeeBus verfolgt das Ziel reale IoT-Szenarien zu emulieren. Die Simulation basiert auf protokollierten Echtzeitdaten von Sensorquellen, muss selber aber nicht in Echtzeit ablaufen. Die Emulationsgeschwindigkeit ist variabel und eine Erhöhung steigert zum Beispiel die Senderate und damit die Netzwerklast. Damit können variable Lastbedingungen erzeugt werden. Die Emulationsumgebung ZeeBus soll verwendet werden, um Protokolle in einem realen Anwendungskontext zu untersuchen. Zum Zeitpunkt der Arbeit konnte bereits mit dem ACDSense-Datensatz experimentiert werden. Dadurch stehen reale Nutzdaten zur Verfügung, die in die Betrachtung mit einfließen. ZeeBus ist im hohen Maße ein Implementationstest von Protokollen respektive Implementionen von diesen Protokollen.

Getestet werden XMPP und MQTT. Beide arbeiten auf TCP. Es werden keine zusätzlichen Auslieferungsgarantien verlangt. Die Wahrscheinlichkeit eines Nachrichtenverlustes wird als gering eingestufft, da der Test im lokalen Gigabit-LAN stattfindet. Durch XML-Streaming verschickt XMPP größere Nachrichten als MQTT und erzeugt damit eine höhere Netzwerklast. Hinzukommen notwendige notwendige Parsing-Operationen. Dies sind die wesentlichen Unterschiede.

ZeeBus kann im aktuellen Stand als Lasttest für Server-, sowie Clientsoftware interpretiert werden. Untersucht wird, wie sich eine Intensivierung der Testparameter auf das System auswirkt, desweiteren, wie sich das Relationsmodell Eins-zu-Viele gegen Viele-zu-Eins verhält. Schlussendlich bietet ZeeBus die Möglichkeit die Implementation der konzeptionellen MQTT-Discovery im realistischen Kontext zu testen.

7.3.2 ABLAUF

Zu Beginn werden die Sender und Ressourcen initiiert. Dazu erstellt die Emulationsumgebung entsprechend der Konfiguration n Senderinstanzen. Für jeden Sender wird abhängig vom Protokoll entweder ein eindeutiger XMPP-Account auf dem Server erstellt, oder ein eindeutiger Name für einen MQTT-Client vergeben. Im Anschluss wird pro Sender eine exklusive Ressource bereit gestellt. Jedem XMPP-Sender wird je ein Multi-User Chat und jedem MQTT-Sender ein Topic zugewiesen. Wenn diese Operationen abgeschlossen sind, können die Empfänger erzeugt werden. Dafür wird eine Discovery nach Ressourcen begonnen. Für jede gefundene Ressource, erzeugt die Anwendungen eine bestimmte Anzahl m von Empfängern, welche sich für die Ressource interessieren. Diese melden sich entweder im Multi-User Chat an oder registrieren sich für ein MQTT-Topic. Bis zu diesem Zeitpunkt wurden noch keine Nachrichten versendet. Erst wenn alle Sender, alle Ressourcen und alle Empfänger erstellt und initiiert sind, kann der Test beginnen. Dieser startet eine Simulation, welcher in Abhängigkeit von Zeitgeschwindigkeit und Simulationsdauer periodisch die Sender dazu veranlasst Nachrichten zu verschicken. Sende- und Empfangszeitpunkt werden gemessen und für eine später Analyse in einer Datei gespeichert. Nach Ender der Simulation melden sich Sender und Empfänger beim Server ab.

7.3.3 AUFBAU

Mit der Simulationsumgebung werden insgesamt 13 Szenarien getestet. Diese unterscheiden sich in Anzahl der Sender, Anzahl zugeordneter Empfänger, Protokoll und Software. Es gibt drei 1:n und drei m:1Relationen, jeweils in den Stufen 1:10, 1:100 , 1:000 und 10:1, 100:1, 1000:1. 1:n bedeutet, dass es einen Sender und genau n Empfänger gibt, die sich für die Ressource interessieren. Im Gegensatz dazu bedeutet m:1, dass es m Sender gibt und für jeden dieser Sender genau einen Empfänger. Getestet wird in einem isolierten LAN. Testrechner und Server sind direkt über einen Gigabit-Switch miteinander verbunden. Als Testrechner wird ein DELL * verwendet. Serverseitig sind OpenFire für XMPP und Mosquitto für MQTT installiert. Als Alternative für MQTT wird zusätzlich im 1000:1 Szenario Apache Apollo verwendet. Der Server ist ein Lenovo ThinkCenter*. Für die Übertragung werden keine Zuverlässigkeitsgarantier konfiguriert, das heißt MQTT wird mit QoS 0 ausgeführt. Die Simulationsumgebung ist auf 500-facher Zeitbeschleunigung eingestellt. Somit sind eine Simulationssekunde 500 Sekunden im Datenzeitraum. Der Simulationszeitraum ist zwischen dem * und dem *.

Bezeichnung	**1:10 MQTT**	**1:100 MQTT**	**1:1000 MQTT**	**10:1 MQTT**	**100:1 MQTT**	**1000:1 MQTT**	**1:10 XMPP**	**1:100 XMPP**	**1:1000 XMPP**	**10:1 XMPP**	**100:1 XMPP**	**1000:1 XMPP**
Sender	1	1	1	10	100	1000	1	1	1	10	100	1000
Empfänger pro Sender	10	100	1000	1	1	1	10	100	1000	1	1	1
Testrechner	DELL, Windows 8											
Server	Lenovo ThinkCentre, OpenFire						Lenovo ThinkCentre, Mosquitto					Lenovo ThinkCentre , Mosquitto/ Apollo
Protokoll	XMPP						MQTT					
Zeitbeschleu -nigung	500x											
Zeitraum	*											
Netzwerk	Isoliertes Gigabit - LAN											
Weiteres	Unzuverlässige Übertragung											

Tabelle 7.34: Testparameter für ZeeBussimulation von ACDSence-Daten.

7.3.4 ERGEBNISSE

Latenz

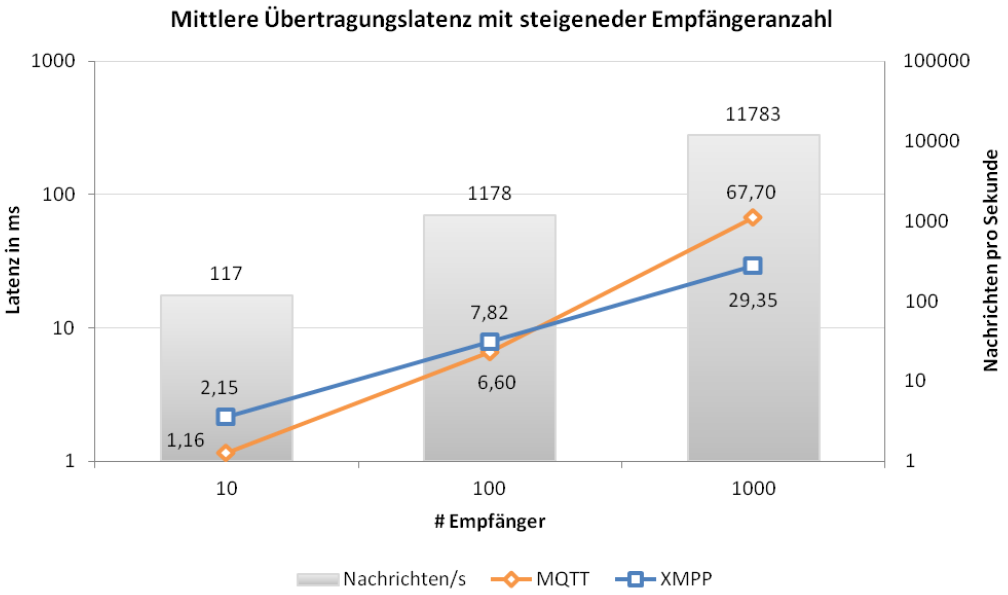


Tabelle 7.35: *

Im ersten Experiment vergleichen wir Eins-zu-Viele Relationen zwischen Sendern und Empfängern. Es gibt einen Sender, eine Ressource und n Empfänger, die sich für diese Ressource interessieren. Ein Sender sendet damit eine Nachricht, welche an n Empfänger ausgeliefert wird. Für n wird in separaten Tests 10, 100 und 1000 gewählt.

Tabelle 7.35 zeigt die Übertragunglatenzen für MQTT und XMPP als Protokoll in Relation zur Nachrichtenrate. Die Nachrichtenrate ist die Aggregation von gesendeten und empfangenen Nachrichten pro Sekunde. Mit steigender Nachrichtenrate erhöht sich für XMPP mittlere Übertragungslatenz. Die Veränderung ist linear, allerdings nicht streng proportional. Ein Verzehnfachung der Nachrichtenrate bewirkt eine 3 bis 4 fach höhere Latenz für XMPP.

Mit steigender Nachrichtenrate erhöht sich ebenso für MQTT die mittlere Übertragungslatenz. Die Veränderung ist nur auf den ersten Blick linear und proportional. Ein Verzehnfachung der Nachrichtenrate bewirkt eine 6 (10 zu 100) bis 10 (100-1000) fach höhere Latenz für MQTT. MQTT zeigt damit einen anwachsenden Anstieg der Latenz. Zusätzliche Testreihen mit feineren Abstufungen in der Anzahl der Empfänger könnten eine ansteigende Kurve in der Latenzentwicklung zeigen.

Bei geringer Nachrichtenrate ist dir durchschnittlich Übertragunsdauer einer Nachricht mit 1,16 Millisekunden rund 1 Millisekunde niedriger, als mit XMPP (2,15 Millisekunden). Im Falle von 100 Empfängern bleibt die Differenz annähernd gleich nur das Verhältnis reduziert sich, da die Latenzen ansteigend am in der Summe unter 10 Millisekunden liegen. Mit einer weiteren Verzehnfachung der Empfängerzahl steigt die Übertragunsdauer bei MQTT auf rund 67 Millisekunden und ist damit mehr als doppelt so hoch, wir bei einer Übertragung mit XMPP (rund 29 Millisekunden).

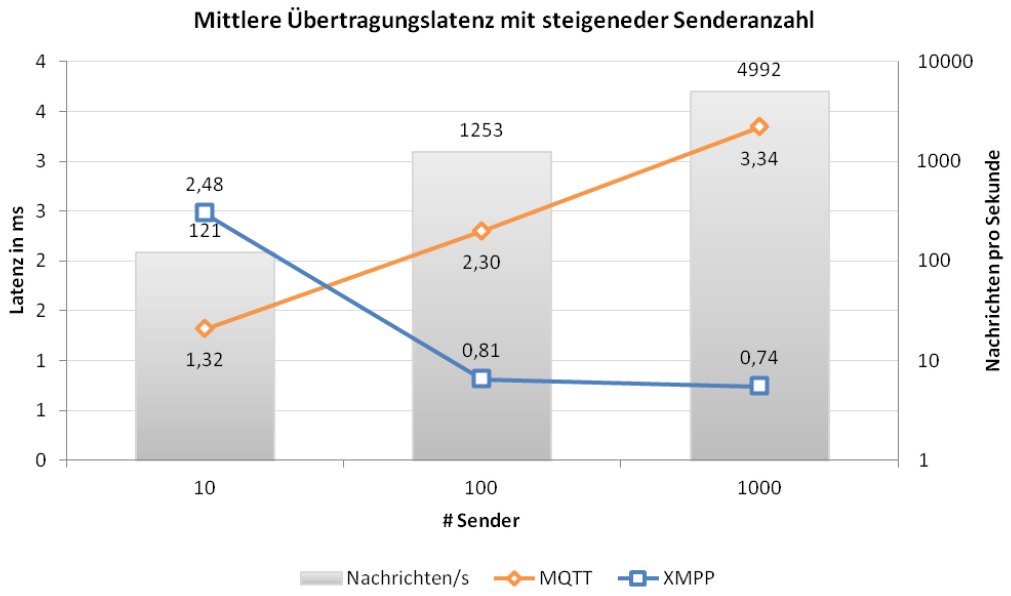


Tabelle 7.36: *

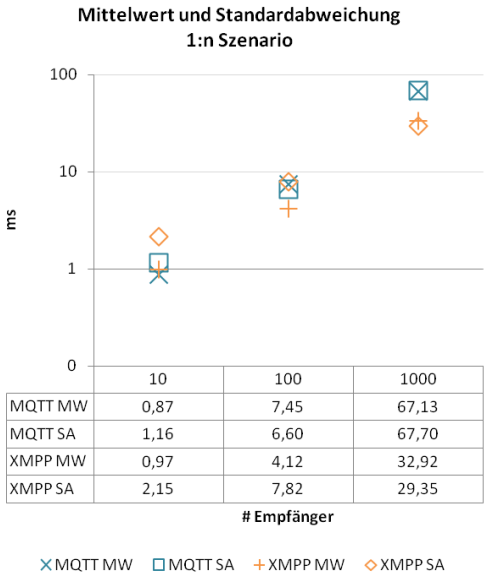
Das folgende Experiment untersucht den Einfluss der Senderanzahl auf die Übertragungslatenz. Für die Viele-zu-Eins Relationen werden Testreihen mit 10, 100, und 1000 Sendern untersucht. Jedem Sender wird eine bestimmte Ressource und jeder Ressource genau ein Emfpänger zu geordnet. Die Sender-Empfänger-Relation ist damit streng genommen Eins zu Eins.

Da in jedem Test die Senderanzahl unterschiedlich ist, ist ebenso der Datensatz für die Simulation verschieden. Daraus resultieren nicht proportionale Verhältnisse hinsichtlich der Nachrichtenrate. Die Nachrichtenrate beträgt bei 100 Sendern 1253 Nachrichten pro Sekunde, was eine Verzehnfachung gegenüber 10 Sendern entspricht und 4992 Nachrichten pro Sekunde bei 1000 Sendern. Dies ist ein Faktor von 4 gegenüber 100 Sendern.

Die mittlere Übertragungslatenz für MQTT erhöht sich linear zur Anzahl der Sender, aber nicht proportional zum Nachrichtenaufkommen. Für jede Multiplikation der Senderanzahl mit 10, wächst die Latenz um 1 Millisekunde an und bleibt in jeder Testreihe unter 4 Millisekunden.

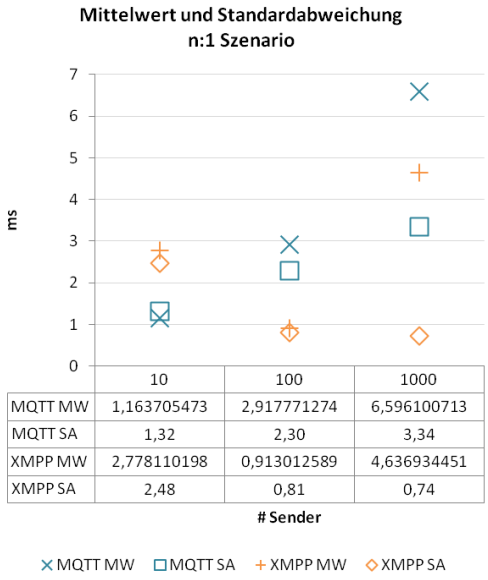
Wird XMPP verwendet sinkt die Übertragungsdauer mit Anzahl der Sender. Beträgt sie bei 10 Sendern noch rund 2,48 Millisekunden, fällt sie auf 0,81 bei 100 Sendern und 0,74 Millisekunden bei 1000 Sendern. Sie fällt damit weder linear noch proportional.

Im Vergleich beider Protokolle übertragt MQTT bei 10 Sendern mit kürzeren Latenzen, bei 100 und 1000 Sendern ist es umgekehrt und der Abstand zwischen den jeweiligen Mittelwerten steigt. Die Werte bleiben in allen Testreihen jedoch unter 4 Millisekunden.



	10	100	1000
MQTT MW	0,87	7,45	67,13
MQTT SA	1,16	6,60	67,70
XMPP MW	0,97	4,12	32,92
XMPP SA	2,15	7,82	29,35

Tabelle 7.37: Standardabweichungen und Mittelwerte von 1:n Szenarien mit XMPP und MQTT.



	10	100	1000
MQTT MW	1,163705473	2,917771274	6,596100713
MQTT SA	1,32	2,30	3,34
XMPP MW	2,778110198	0,913012589	4,636934451
XMPP SA	2,48	0,81	0,74

Tabelle 7.38: Standardabweichungen und Mittelwerte von n:1 Szenarien mit XMPP und MQTT.

Die Standardabweichung ist ein Maß für die Streuung von Werten um den Mittelwert. In Übertragungsnetzen kann die Standardabweichung zur Beschreibung von Schwankungen in der Latenzzeit (Jitter) genutzt werden. Tabelle 7.37 zeigt die Standardabweichung in Kombination mit dem Mittelwert für Tests mit einem Sender und vielen Empfängern. Die Abbildung zeigt deutlich, dass Standardabweichung und Mittelwert korrelieren.

Dieses Verhalten ist für Viele-zu-Eins Relationen geringer ausgeprägt, wie in Tabelle 7.38 zu sehen.

Overhead

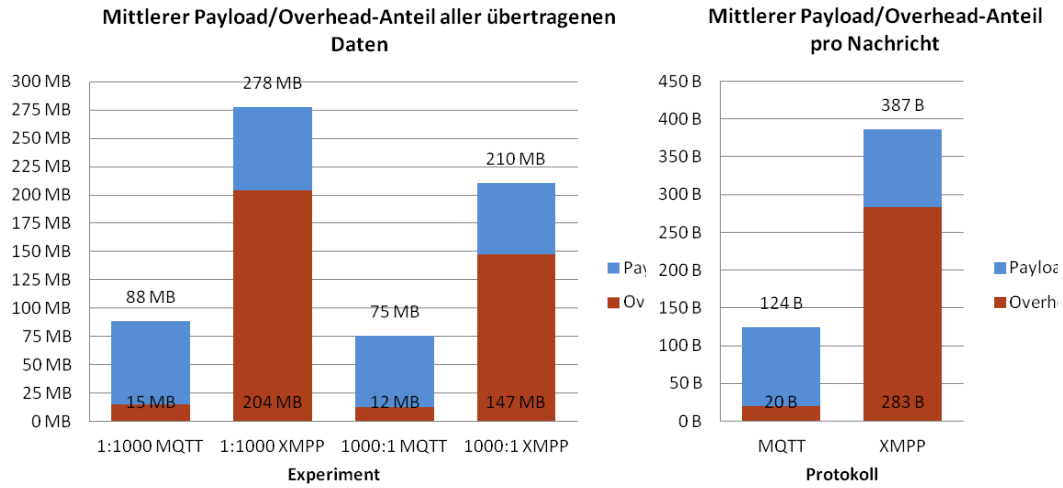


Tabelle 7.39: *

Beim Senden und Eintreffen von Nachrichten wird der Payload sowie die Größe der Nachricht in Bytes protokolliert. Daraus lässt sich errechnen, wie viel minimale Last die Protokolle im Netzwerk erzeugen. Der Wert ist die minimale Last, da durch darunterliegende Protokolle, wie zum Beispiel TCP zusätzliche Pakete bei der Übertragung erzeugt werden. Von Interesse ist insbesondere das Verhältnis von

Overhead zu Nutzlast. Der Overhead sind Informationen, die ein Protokoll zur Transportsteuerung benötigt. Die Nutzlast/der Payload ist die eigentliche zu übertragende Information. Die Werte sind auf ganze Megabyte gerundet.

Bei dem vorherigen Test mit 1 Sender und 1000 Empfänger wurden in der Summe mit MQTT 88 Megabyte und mit XMPP 278 Megabyte übertragen. Damit erzeugt XMPP auf Anwendungsebene mehr als das 3-fache (3,16) an Datenvolumen. In beiden Fällen beträgt das Nutzlastvolumen 74 Megabyte. Der Anteil der Nutzlast am Gesamtvolumen beträgt bei MQTT 83 Prozent und bei XMPP 26 Prozent. Mit 204 Megabyte XMPP-Overhead ist allein dieser Anteil größer, als das gesamte Datenvolumen von MQTT. Insgesamt wurden 707707 Nachrichten übermittelt.

Ähnlich verhält es sich bei umgekehrter Relation mit 1000 Sender und einem Empfänger. Die Anzahl gesendeter Nachrichten ist mit 599072 geringer. Im Verhältnis ist das Datenaufkommen von XMPP und MQTT mit 210 Megabyte zu 75 Megabyte weniger als 3-mal (2,8) so hoch. Mit beiden Protokollen wurden 63 Megabyte Nutzdaten übertragen. Damit beträgt der Anteil am Gesamtvolumen bei MQTT 84 Prozent und bei XMPP steigt er auf 30 Prozent. Mit 147 Megabyte ist allein der Protokolloverhead von XMPP knapp doppelt so hoch, wie das gesamte durch MQTT erzeugt Datenvolumen.

Für eine durchschnittliche Nachricht verhält es sich wie folgt. Der Payload beträgt 104 Byte. MQTT übertragt diesen mit einem Header von 20 Byte. Damit ist eine MQTT-Nachricht im Schnitt 124 Byte groß. Eine XMPP-Nachricht fasst dagegen 387 Byte. Damit beträgt der Overhead 283 Byte. Dieser ist mehr als doppelt so groß, wie eine typische MQTT-Nachricht. Im Schnitt sind 84 Prozent der mit MQTT übertragenden Daten Nutzdaten, bei XMPP sind es 27 Prozent. XMPP erzeugt 3-mal so große Nachrichten, wie MQTT.

MQTT-Broker

Bei nichtfunktionalen Leistungstests ist es wichtig zu verstehen, dass ein Konzept immer nur so gut ist, wie seine Umsetzung. Die Eigenschaften eines Broker-basiertes Protokolls sind in einem hohen Maße abhängig von der Brokerimplementierung. Bisher kam Mosquitto als MQTT-Broker zum Einsatz. Mosquitto nutzt eine Single-Thread Architektur. Das impliziert zum einen, dass alle Operationen sequentiell abgearbeitet werden und zum anderen dass nur die Leistung eines Rechnerkerns zur Verfügung steht.

Die ACDSense-Emulation ist im Prinzip ein Lasttest, der eine hohe Anzahl von Nachrichten zum gleichen Zeitpunkt erzeugt. Ein sequentiell arbeitender Broker wird daher alle eintreffenden Nachrichten sequentiell bearbeiten. Die Annahme ist nun, dass erstens, sich durch die sequentielle Behandlung die Verarbeitungslatenz auf Brokerseite aggregiert und zweitens, ein Broker, der parallel mehrere Kerne nutzt, Nachrichten effizienter bearbeiten kann.

Für eine Bewertung der Annahme wird Mosquitto durch einen weiteren MQTT-Broker (Apache Apollo) ausgetauscht. Dieser arbeitet mit mehreren Threads.

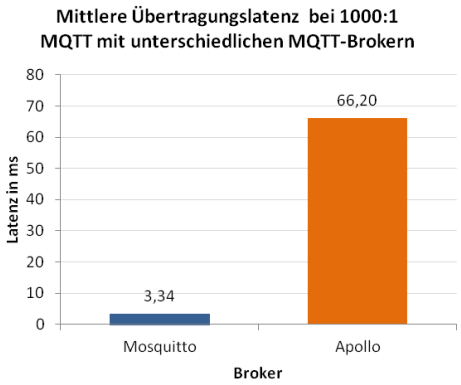


Tabelle 7.40: Durchschnittliche Übertragungslatenz mit MQTT bei 1000 Sendern und 1 Empfänger und verschiedenen MQTT-Brokern.

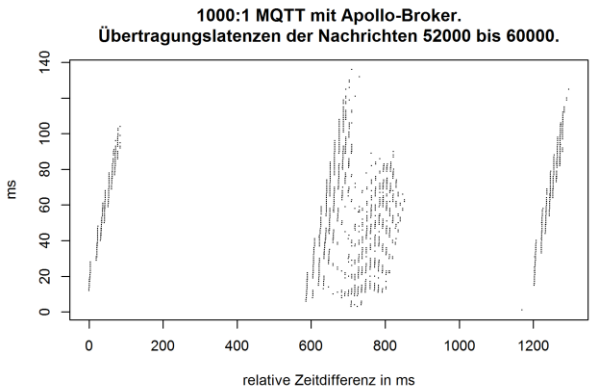


Tabelle 7.41: Latenzen für einzelne Nachrichten mit Apollo als MQTT-Broker. Diagramm zeigt Latenzen der zum Zeitpunkt x eingetroffen Nachrichten.

Wir testen Apollo mit 1000 Sendern mit je einem Empfänger und stellen die Mittelwerte gegenüber. Aus Tabelle 7.40 ist zu entnehmen, dass der Austausch des Brokers keine Verbesserungen brachte. Im Gegenteil, durchaus überraschend ist der gemessene Mittelwert mit rund 66,2 Millisekunden 20 Mal höher als bei Mosquitto mit 3,34 Millisekunden. Die Annahme konnte sich nicht bestätigen.

Ein Blick in den Latenzverlauf zeigt das Auslieferungsverhalten von Apollo. In Tabelle 7.41 sind die Latenzen für die Nachrichten Nummer 52.000 bis 60.000 dargestellt. Die X-Achse gibt den Empfangszeitpunkt einer Nachricht relativ zur ersten betrachteten Nachricht an, die Y-Achse die Latenz der zu diesem Zeitpunkt empfangen Nachricht. Wenn man die ersten der drei Punktmengen betrachtet, dann treffen die Letzten Nachrichten ungefähr nach 100 Millisekunden ein. Diese haben eine Latenz von ebenfalls 100 Millisekunden. Damit wurden sie zum gleichen Zeitpunkt abgesendet, wie die erste betrachtete Nachricht. Betrachtet man den Verlauf bis dahin, erkennt man eine gestufte lineare Entwicklung. Somit haben alle bis zur 100 Millisekunde eingetroffen Nachrichten den gleichen Versendezeitpunkt. Ähnliches gilt für die dritte Punktmenge. Die Linearität in den Latenzen ist ein Indiz für eine sequentielle Abarbeitung, trotz Multi-Thread-Architektur. Die mittlere Punktmenge zeigt ein anderes Verhalten. Zwar treten hier ebenso lineare Latenzstufen auf, allerdings parallel zueinander.

Der Austausch des Brokers ergab keine besseren Messwerte. Im Gegenteil, die mittlere Latenz verschlechtert sich bei Apollo signifikant. In den Diagrammen wird eine parallele Verarbeitung auf Broker-Seite erkennbar. Anscheinend ist diese nicht effizient umgesetzt, zumindest nicht effizient im Sinne einer minimalen Übertragungslatenz.

7.3.5 Zusammenfassung

- Allgemein
  - Keine Nachrichtenverluste feststellbar
  - Discovery funktioniert
  - ID funktioniert
- Latenz
  - MQTT bei wenigen Knoten leicht besser als XMPP
  - Mit steigender Knotenanzahl verschiebt sich das Verhältnis zu Gunsten von XMPP
  - XMPP bei 1000 Knoten besser
    - Deutlich bei 1:1000 Relation

        - Deutlich weniger bei 1000:1
    - Abweichungen im Millisekundenbereich inkonsitent wegen Uhrauflösung
    - 1000:1 Relation liefert bessere Werte
    - 1:1000 linearer Latenzanstieg mit Knotenanzahl erkennbar
    - Grund: 1:1000 ein Event erzeugt 1000 Nachrichten in einem Zeitpunkt
        - Höhere Peaklasten in diesem Szenario
            - Senderate herziehen
        - Verdopplung notwendig
        - Auf Brokerseite?
        - Singlethreadbroker
            - Sequentielle abarbeitung
        - Queue versopfung
        - Netzwerkinterfacebuffer
- Volumen
    - XMPP deutlich mehr Datenlast und Overhead
        - Overhead ist 14 mal so hoch, wie bei MQTT
    - Datenlast scheint sich nicht auf Übertragung ausgewirkt zu haben
    - Für mobile Verbindungen äußertst kritisch
    - MQTT kompakter header
    - Verhältnis verschlechtert sich mit Datenoptmierung
    - Typestrukturierung und Daten von Stuktur Trennung

8 Zusammenfassung Und Ausblick

In den Grundlagen ging es darum, Kriterien zusammenzutragen, die für die Bewertung einer Kommunikation entscheidend sind. Darüber hinaus erfolgte eine erste Zusammenfassung verbreiteter Protokolle, die gemeinsam mit dem Internet der Dinge genannt werden. Ergänzt wurden die Grundlagen durch Charakteristiken der Netzwerkschicht, insbesondere drahtloser Schnittstellen und einem Querschnitt des überaus umfangreichen Themas der *Quality of Service*. Zusammenfassend kann zwischen präventiven und reaktiven Verfahren unterschieden werden, um Dienstgütegarantien anzubieten. Die aktuellen Umsetzungen setzen entweder auf eine direkte Reservierung von Ressourcen oder Kennzeichnen den Netzwerkverkehr mit bestimmten Attributen für eine systematisierte Behandlung.

Darauf aufbauend hinterließ die Recherche nach verwandten Arbeiten einen eher ambivalenten Eindruck. Es existiert eine Vielzahl von Untersuchungen auf Netzwerkebene, bei denen sehr spezielle Anpassungen und Optimierungen vorgenommen werden. Das Potential, dass sich diese Verbesserungen in einen heterogenen, komplexen Netz und Netzkombination etablieren, wird entgegen als gering eingeschätzt. Auf Transportebene dominieren UDP und TCP. Beide unterscheiden sich deutlich in ihren Fähigkeiten der Transportsteuerung und verhalten sich daher Grund auf verschieden in Netzen. Beide sind nicht für drahtlose Verbindungen angepasst. Es fehlen fundamentale Fähigkeiten, den Anforderungen mobiler Kommunikation gerecht zu werden. Nicht wenige Studien erkennen in SCTP ein Transportprotokoll, dass diese Kluft in Zukunft verringern kann. Allerdings weist SCTP ebenso Nachteile auf, in erster Linie einen ungeeigneten Umgang mit dynamischen IP-Änderungen. mSCTP versucht dieses Mängel zu beheben. Auf Anwendungsebene finden sich nur sehr wenige Studien, die den Protokollen dieser Schicht eine umfassende analytische Betrachtung widmen. Allein DDS und CoAP weise eine Reihe von Untersuchungen, oftmals innerhalb lokaler, homogener Netze, auf.

Eine weitere Erkenntnis ist, erstens die mangelnde Bereitstellung von netzinternen QoS-Mechanismen nach außen und folglich, zweitens eine unzureichende Unterstützung von Übertragungsgarantien über Netzwerkgrenzen hinweg. Insbesondere sind Mobilfunknetze abgeschottete Bereiche für Endteilnehmer und -geräte. Nur mit einer Aushandlung und Bereitstellung von Dienstgütegarantien über administrative Domänen hinweg ist eine echte Ende-zu-Ende-Semantik zu realisieren.

Der Mehrwert der Analyse besteht darin nun an zentraler Stelle einen umfangreichen Überblick und Vergleich über die funktionalen Eigenschaften und individuellen Charakteristiken der Protokolle anbieten zu können. Dafür wurden IoT-Szenarien entworfen und dementsprechend seziert. Ein Protokoll ist immer in seinen Anwendungskontext zu bewerten. Umgekehrt resultiert daraus, dass die Anwendung die Bewertung beeinflusst. Ferner ist ein Anwendungsprotokoll eine Kombination aus Entitäten der darunter liegenden Schichten. Zum Beispiel nutzt MQTT für den Transport TCP und TCP verschickt IP-Paketen über eine konkrete Netzwerkschnittstelle. Das heißt auch, dass die einzelnen Schichten die Eigenschaften der darunter liegenden Schichten erben. TCP zeigt bisweilen ein kritisches Verhalten in mobilen eingeschränkten Netzwerken. Diese Nachteile übertragen sich damit auf MQTT. Die Bewertung eines Protokoll muss daher schichtübergreifend erfolgen. Darüber hinaus wirkt sich eine Veränderung auf einer unteren Schicht auf das Verhalten und damit die Bewertung der obersten Schicht aus.

Dabei unterscheiden die Eigenschaften der Netzwerktechniken signifikant. Das drahtgebundene Kernnetz überträgt Daten mit hoher Geschwindigkeit und Zuverlässigkeit. Bei Funknetzen dagegen kommt es zu Schwankungen in der Übertragungsparametern bis hin zu Verbindungsunterbrechungen. Paketverluste und längere Übertragungszeiten sind die Folge. Hinzu kommen dynamische Wechsel der Zugangspunkte/Funkzellen, die eine Änderung der IP-Adresse zur Folge haben können. Trotz dieser offensichtlichen Herausforderungen bieten die betrachteten Protokolle keinen Zugriff auf die tieferen Ebenen. Eine Konfiguration, geschweigende ein Austausch des Transportprotokolls ist nicht möglich. Auch Eigenschaften und Ereignisse der Netzwerkebene bleiben verborgen.

Hinsichtlich der Anforderungen, die das Internet der Dinge bestimmt, ist eine der wichtigsten, die möglich Ressourcen und Teilnehmer im Kommunikationsnetz dynamisch zu finden. Nicht alle Protokolle spezifizieren Discovery-Mechanismen. Für MQTT und AMQP müssen auf Anwendungseben alternativen bereit gestellt werden. Ferner unterscheiden sich die Mechanismen in zentrale dienstbasierte und dezentrale Multicast-gesteuerte, wobei letztere zu einem unvorhersehbaren Nachrichtenaufkommen im Netz führen kann und eine Discovery über Netzwerkgrenzen hinweg aufwendig zu realisieren ist. Alle Protokolle implementieren eine vom Transportprotokoll unabhängige zuverlässige Übertragung. Von allen bietet DDS den umfangreichsten Katalog an QoS-Parametern an. XMPP hebt sich durch seine native Erweiterbarkeit ab. Diese führt allerdings zu einem deutlich größeren Overhead und erzeugt bisweilen Probleme hinsichtlich der Interoperabilität zu anderen Implementierungen. CoAP besticht dank hybrider Topologie durch Flexibilität, zeigt jedoch Schwierigkeiten mit größeren Datenmengen. MQTT ist einfach und kompakt. Eine Schwachstelle ist dessen Single-Broker-Architektur. Der Fokus von AMQP liegt weniger auf Effizienz, sonder auf die Zuverlässigkeit des Systems. AMQP wird daher Protokoll für die Server-zu-Server-Kommunikation betrachtet.

Zur Analyse gehörte ebenso die Erfassung von Merkmalen periodischer Datenübertragung. Da es sich im Kern dabei um wiederkehrende Ereignisse handelt, werden daraus Rückschlüsse über den Bedarf an Ressourcen möglich. Mit der Annahme, dass sich die übertragenen Informationen ähneln, kann die benötigte Bandbreite geschätzt werden, um schlussendlich konkrete Reservierungen einzuleiten. Die Szenarien lassen vermuten, dass zum einen geringe Datenmengen übertragen werden und zwischen den Übertragungsperioden lange Ruhephase existieren. Ferner werden nicht selten Daten ähnlicher Struktur übermittelt. In einen solchen Fall, ist es sinnvoll Struktur und Daten voneinander zu trennen. Kombiniert mit einer Typisierung kann die Datenmenge drastisch reduziert werden. Da die Datenerfassung, nicht unbedingt mit der Datenübermittlung zeitlich konstant erfolgen muss, sind Methoden der Datenbündelung notwendig. Ein Empfänger kann sich alternativ zum aktuellen Wert für die Folge von Werten oder die Interpretation von Werten interessieren. Nicht zuletzt ist zwischen einem periodischen Senden und einem periodischen Empfangen zu unterscheiden.

Im Implementationskapitel ging es nicht primär um die Darstellung der geleisteten Programmierarbeit. Im Vordergrund standen Einschränkungen, für die ein Lösung gefunden werden musste, um einen erfolgreichen Testablauf zu garantieren. Bis zu dem Zeitpunkt existierte für ACDSense ein Protokollunterstützung für XMPP. Diese wurde um MQTT erweitert. Dafür musste zum einen eine Methode entwickelt werden, um Nachrichten systemweit eindeutig zu identifizieren. Zum anderen fehlte es an einem Mechanismus zum Erfassen von bestehenden Topics. Dafür wurde eine Domänenmodell vorgestellt, dass es ermöglicht ohne einen zusätzlichen Dienst, Topics auf einem MQTT-Broker zu registrieren und abzufragen. Die Lösung zur Generierung von Nachrichten-IDS ist ein MD5-Hashwertes aus der

Kombination von Topic und Nachricht. Für das Szenario reicht dies aus, um eindeutige Nachrichten-IDs zu erzeugen. Da diese Kombination nicht in jedem Falle eindeutig ist (bei identischen Nachrichten), sollte anstatt der Nachricht selber, deren MQTT-interne Nachrichten-ID verwendet werden. Diese ist zwar nur innerhalb einer Client-Server-Relation eindeutig, aber in Kombination mit der Client-ID (die nur einmal pro Server existiert) wird eine globale Eindeutigkeit erreicht. Der dritte wichtige Aspekt betrifft die zeitliche Synchronisierung von getrennten Rechnern. Diese ist notwendig, um bei der Testauswertung korrekte Übertragungslatenzen berechnen zu können. Da die Systemuhren untereinander abweichen, müssen sie über das Netzwerk angepasst werden. Diese Anpassung ist über NTP möglich. Die notwendigen Instruktionen sind im Kapitel beschrieben.

Der Hauptteil der Arbeit zum einen die Analyse zum anderen die Evaluation der Protokolle. Diese wurde zugleich genutzt, um dir vorher konzipierte Zeitsynchronisation zu testen. Das passende ist so gewählt, dass es konstante Testergebnisse liefern und anderweitige Einflüsse vermeiden. Die Ergebnisse zeigen, dass NTP zur Uhrensynchronisation im drahtgebunden LAN geeignet ist und eine Genauigkeit von rund einer Millisekunde erreicht werden kann. Bevor eine Test mit aktiver NTP-Synchronisierung gesteuert wird, solle eine gewisse Zeit für den Abgleich der Uhren eingeplant werden.

Aus der Analyse der periodischen Daten geht hervor, dass zwischen den Senderperioden lange Ruhephasen auftreten können. Auf ressourcenschwachen Systemen mit mobiler Energieversorgung, kann es durchaus sinnvoll sein, während dieser Ruhephase eine Verbindung zu vermeiden beziehungsweise zu beenden. Das impliziert, dass vor dem Senden erst eine neue Verbindung aufgebaut werden muss, zumindest bei verbindungsorientierten Protokollen. Diese Prozedur dauert bei den Protokollen unterschiedlichen. AM deutlich längsten bei XMPP, dass über 100 Millisekunden, MQTT dagegen nur wenige Millisekunden benötigt. MQTT und CoAP liegen damit im drahtgebunden Gigabit-LAN gleicht auf, obwohl CoAP ein verbindungslosen Protokoll ist. Eine Veränderung der QoS-pArameter zeigte keinen Einfluss auf die Dauer. Des Weiteren wurde die Übertragungslatenz einer einzelnen Nachricht gemessen. Für alle Protokolle und unzuverlässiger Übertragung lag diese im einstelligen Millisekundenbereich. Eine garantierte Auslieferung mit MQTT zeigt eine Erhöhung, mit CoAP keine Erhöhung der Auslieferungszeit.

Das ACDSense-Szenario ist eine geeignete Umgebung zum Testen von Protokollimplementationen. Ein wesentliches Merkmal ist die Emulation von realen Kommunikationsströmen, in dem vorher aufgezeichnete Wetterdaten über eine variable Simulation im Netz verteilt werden. Durch die Ehöhung der Sender- oder Empfängeranzahl sowie der justierbaren Geschwindigkeit sind Skalierungs- und Effizienz-Test und Lastbedingungen möglich. Für Eins-zu-Viele-Relationen ergab sich, dass sowohl für MQTT, als auch für XMPP die durchschnittliche Latenz mit Anzahl der Empfänger steigt und XMPP geringere Latenzen erzeugt. Ferner korreliert die Standartabweichung mit dem Mittelwert. In Viele-zu-Eins-Relationen fielen die Übertragungslatenzen deutlich geringer aus. Diese stiegen nur leicht mit steigenden Senderanzahl an. XMPP und MQTT zeigten ähnliche Messwerte und die Korrelation zwischen Mittelwert und Standartabweichung ist deutlich geringer. Die beiden unterschiedlichen Relationsverhältnisse sind aber im Grunde nicht direkt miteinander vergleichbar. Weder sind die Nachrichten die gleichen, noch ist die Nachrichtenanzahl identisch. Auch erzeugen die Testvarianten unterschiedlich Senderaten und damit Lastspitzen während der Übertragung (Senderaten abbilden*). Neben der Gegenüberstellung von XMPP und MQTT, wurde aufgrund der unterschiedlichen Thread-Architektur für MQTT ein alternativer Broker getestet. Dadurch erhöhten sich die Latenz signifikant und

verdeutlich, dass die Leistungsfähigkeit eines Protokoll maßgeblich von dessen implementierten Komponenten abhängig ist.

In allen Testreihen sind bei keinem Protokoll Nachrichtenverluste aufgetreten

Ein Thematik wurde von vorne herein bei der Betrachtung ausgelassen, nämliche Sicherheitskritischen Aspekte. Sicherheit ist hier im Kontext von vertrauenswürdigen Daten, Authentifizierung und Autorisierung zu verstehen. Sicherheit ist an für sich ein komplexes Thema und für Sicherheit gilt ebenso, dass sie im Anwendungskontext zu bewertet ist, je nachdem, welche Anforderungen ein gegebenes Szenario verlangt. Die Protokolle unterstützen diesbezüglich unterschiedliche Mechanismen.

Ein ressourcenarmes Gerät ist nicht selten mit einer schwächeren CPU ausgestattet. Umso wichtiger wird es die Kommunikation leichtgewichtig zu gestalten, um so wenige Operationen wie nötig auszuführen. Damit kann gleichzeitig der Energiebedarf gesenkt werden. Um dieses Kriterium bewerten zu können ist es sinnvoll die ACDSense-Testumgebung um Möglichkeiten erweitern, den Speicherverbrauch sowie die CPU-Auslastung zu messen. Diesbezüglich sollten folgende Hinweise beachtet werden:

Eines der größten Hindernisse, wie während der Analyse aufgetaucht sind, ist der fehlende Zugang zu unteren Schichten. umschreibt diese Fähigkeit als *Underlay Awarness **. Mit dem Wissen über konkrete Netzwerkeigenschaften, insbesondere in mobilen Netzen, könnte adäquat auf Schwankungen, Verbindungsabbrüchen und IP-Änderungen reagiert werden. Erst mit dem Zugriff auf netzabhängige QoS-Mechanismen und der Etablierung über Netzwerkgrenzen sind echte Ende-zu-Ende-Garantien möglich.

Periodische Daten besitzen charakteristische Merkmale. Nicht zuletzt die im Analyseteil vorgestellte Trennung von Daten und Struktur verdeutlich das Potential einer individuellen Behandlung durch bereitgestellt Komponenten. Dieser datenzentriete Ansatz kann im hohen Maße unabhängig vom Protokoll realisiert und angeboten werden.

Bei allen Testergebnissen ist es notwendig darauf hinzuweisen, dass die Test selber in einem drahtgebunden lokalen Netzwerk stattfanden. Zum einen vermeidet man damit ungewollte Einflüsse andere Ströme und Teilnehmer, zum anderen ist ein solches Netz ein synthetisches, ideales Netz. Für eine vollständige Untersuchung, vor allem mobiler Verbindungen, müssen solche Verbindungen geschaffen werden, weil sie ein Protokoll signifikant beeinflussen. Die Implementierung einer mobilen Umgebung stellt keine triviale Aufgabe dar. Der Aufbau einer kabellosen Netzstruktur führt zu den bekannten Schwankungen in den Übertragungsparametern. Dadurch ändern sich die Bedingungen in jedem Testdurchlauf. Die Testergebnisse sind daher nur bedingt miteinander vergleichbar. Die Simulation einer Funkübertragung ist dagegen nur so gut, wie das Modell, das sie definiert.

LITERATURVERZEICHNIS

[00a] Electronic Healthcare - First International Conference, eHealth 2008, London, September 8-9, 2008,

[00b] jabber.org - the original XMPP instant messaging service. URL http://www.jabber.org/. - abgerufen am 2014-04-08

[00c] MQTT used by Facebook Messenger | MQTT. URL http://mqtt.org/2011/08/mqtt-used-by-facebook-messenger. - abgerufen am 2014-04-09

[00d] OData Version 3.0 Core Protocol. URL http://www.odata.org/documentation/odata-version-3-0/odata-version-3-0-core-protocol/. - abgerufen am 2014-04-10. — Open Data Protocol

[00e] IoT Comic Book. URL http://iotcomicbook.org/. - abgerufen am 2014-06-25. — IoT Comic Book

[00f] End-to-End Quality of Service Over Heterogeneous Networks

[00g] What can DDS do for You?, Twin Oaks Computing, Inc

[00h] joachimlindborg/XMPP-IoT. URL https://github.com/joachimlindborg/XMPP-IoT. - abgerufen am 2014-06-26. — GitHub

[00i] [XML] mqtt-avahi-service-descriptor - Pastebin.com. URL http://pastebin.com/0RhJyZud. - abgerufen am 2014-06-08. — Pastebin

[07] Data Distribution Service (DDS). URL http://www.omg.org/spec/DDS/1.2/PDF/. - abgerufen am 2014-04-09. — OMG Object Management Group

[08] Robust Java benchmarking, Part 1: Issues. URL http://www.ibm.com/developerworks/java/library/j-benchmark1/index.html#table1. - abgerufen am 2014-06-29

[10a] MQ Telemetry Transport (MQTT) V3.1 Protocol Specification. URL http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html. - abgerufen am 2014-04-09

[10b] The Real-Time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol (DDSI-RTPS). URL http://www.omg.org/spec/DDSI-RTPS/2.1. - abgerufen am 2014-04-09. — OMG Object Management Group

[12] GData Protocol. URL https://developers.google.com/gdata/docs/2.0/basics. - abgerufen am 2014-04-10. — Google Data APIs

[13] System timers granularity. URL https://code.google.com/p/javasimon/wiki/SystemTimersGranularity. - abgerufen am 2014-06-29

[14a] Communications protocol. Wikipedia, the free encyclopedia.

[14b] Performance evaluation of MQTT and CoAP via a common middleware. In: : IEEE, 2014 — ISBN 978-1-4799-2843-9, 978-1-4799-2842-2, S. 1–6

[AlSh09] ALDASOUQI, IYAD ; SHAOUT, ADNAN: Earthquake Monitoring System Using Ranger Seismometer Sensor. In: International Journal of Geology (2009), Nr. 1

[AnMO12] ANDO, REMI ; MURASE, TUTOMU ; OGUCHI, MASATO: TCP and UDP QoS Characteristics on Multiple Mobile Wireless LANs. In: Proc. the 35th IEEE Sarnoff Symposium, 2012

[BaAL06] BAI, HAOWEI ; ATIQUZZAMAN, MOHAMMED ; LILJA, DAVID J.: Layered view of QoS issues in IP-based mobile wireless networks. In: International Journal of Communication Systems Bd. 19 (2006), Nr. 2, S. 141–161

[BaBh13] BANDYOPADHYAY, SOMA ; BHATTACHARYYA, ABHIJAN: Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. In: : IEEE, 2013 — ISBN 978-1-4673-5288-8, 978-1-4673-5287-1, 978-1-4673-5286-4, S. 334–340

[BaRa14] BARBARA HARTEL ; RAM JEYARAMAN: OData Version 4.0 Part 1: Protocol. URL http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html. - abgerufen am 2014-04-10. — OASIS - Advancing open standards for the information society

[BCSS12] BAZZANI, MARCO ; CONZON, DAVIDE ; SCALERA, ANDREA ; SPIRITO, MAURIZIO A. ; TRAINITO, CLAUDIA IRENE: Enabling the IoT Paradigm in E-health Solutions through the VIRTUS Middleware. In: : IEEE, 2012 — ISBN 978-1-4673-2172-3, 978-0-7695-4745-9, S. 1954–1959

[BFLM11] BLUM, N. ; FIEDLER, J. ; LANGE, L. ; MAGEDANZ, T.: Application-driven Quality of Service for M2M communications. In: 2011 15th International Conference on Intelligence in Next Generation Networks (ICIN), 2011, S. 41–45

[BiBi02] BICSI ; BICSE: Network Design Basics Cabling Professionals. New York : Mcgraw Hill Book Co, 2002 — ISBN 9780071399166

[BSSS13] BENDEL, S. ; SPRINGER, T. ; SCHUSTER, D. ; SCHILL, A. ; ACKERMANN, R. ; AMELING, M.: A service infrastructure for the Internet of Things based on XMPP. In: 2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013, S. 385–388

[ChLK08] CHEN, GUANLING ; LI, MING ; KOTZ, DAVID: Data-centric Middleware for Context-aware Pervasive Computing. In: Pervasive Mob. Comput. Bd. 4 (2008), Nr. 2, S. 216–253

[Chri00] CHRISTOPH FINOTTO: OData - Leichtgewichtiger Zugri_ auf Businessdaten in mobilen Anwendungen

[CoFN10] CORRADI, A. ; FOSCHINI, L. ; NARDELLI, L.: A DDS-compliant infrastructure for fault-tolerant and scalable data dissemination. In: 2010 IEEE Symposium on Computers and Communications (ISCC), 2010, S. 489–495

[DSES11] DWORAK, A. ; SOBCZAK, M. ; EHM, F. ; SLIWINSKI, W. ; CHARRUE, P.: Middleware Trends And Market Leaders 2011. URL http://cds.cern.ch/record/1391410. - abgerufen am 2014-06-26. — Conf. Proc.

[DVAA13] DAVID, LINCOLN ; VASCONCELOS, RAFAEL ; ALVES, LUCAS ; ANDRÉ, RAFAEL ; ENDLER, MARKUS: A DDS-based middleware for scalable tracking, communication and collaboration of mobile nodes. In: Journal of Internet Services and Applications Bd. 4 (2013), Nr. 1, S. 1–15

[ElSa12] EL-ATAWY, A. ; SAMAK, T.: End-to-end verification of QoS policies. In: 2012 IEEE Network Operations and Management Symposium (NOMS), 2012, S. 426–434

[EsCG09] ESPOSITO, CHRISTIAN ; COTRONEO, DOMENICO ; GOKHALE, ANIRUDDHA: Reliable Publish/Subscribe Middleware for Time-sensitive Internet-scale Applications. In: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS '09. New York, NY, USA : ACM, 2009 — ISBN 978-1-60558-665-6, S. 16:1–16:12

[Espo00] ESPOSITO, CHRISTIAN: Data Distribution Service (DDS) Limitations for Data Dissemination w.r.t. Large-scale Complex Critical Infrastructures (LCCI). In: : Diparimento di Informatica e Sistemistica (DIS), Univerita degli studi di Napoli Federico

[FeMe11] FETTE, I. ; MELNIKOV, A.: The WebSocket Protocol, Request for Comments : IETF, 2011. — Published: RFC 6455 (Proposed Standard)

[GoKa13] GOYAL, RAMAN KUMAR ; KAUSHAL, SAKSHI: A Survey of mSCTP for Transport Layer Mobility Management. In: Journal of Advances in Information Technology Bd. 4 (2013), Nr. 1, S. 20–27

[Gord07] GORDON HUNT: DDS Advanced Tutorial: Using DDS QoS to Solve Real-world Problems (2007)

[GrHo07] GREGORIO, J. ; HORA, B. DE: The Atom Publishing Protocol, Request for Comments : IETF, 2007. — Published: RFC 5023 (Proposed Standard)

[GSBV03] GIORDANO, S. ; SALSANO, S. ; VAN DEN BERGHE, S. ; VENTRE, GIORGIO ; GIANNAKOPOULOS, D.: Advanced QoS provisioning in IP networks: the European premium IP projects. In: IEEE Communications Magazine Bd. 41 (2003), Nr. 1, S. 30–36

[HMES08] HILDEBRAND, JOE ; MILLARD, PETER ; EATMON, RYAN ; SAINT-ANDRE, PETER: Service Discovery. URL http://xmpp.org/extensions/xep-0030.html. - abgerufen am 2014-06-19. — This specification defines an XMPP protocol extension for discovering information about other XMPP entities. Two kinds of information can be discovered: (1) the identity and capabilities of an entity, including the protocols and features it supports; and (2) the items associated with an entity, such as the list of rooms hosted at a multi-user chat service.

[HuGa04] HUANG, YONGQIANG ; GARCIA-MOLINA, HECTOR: Publish/Subscribe in a Mobile Environment. In: Wirel. Netw. Bd. 10 (2004), Nr. 6, S. 643–652

[Isod14] ISODE: Isode XMPP Whitepapers. URL http://www.isode.com/whitepapers/white-xmpp.html. - abgerufen am 2014-06-25. — isode

[JiAn10] JI, ZHANG ; ANWEN, QI: The application of internet of things(IOT) in emergency management system in China. In: 2010 IEEE International Conference on Technologies for Homeland Security (HST), 2010, S. 139–142

[KeSB06] KEUN JAE LEE ; SANG SU NAM ; BYUNG IN MUN: SCTP efficient flow control during handover. In: : IEEE, 2006 — ISBN 1-4244-0269-7, S. 69–73

[KGKS11] KLAUCK, R. ; GAEBLER, J. ; KIRSCHE, M. ; SCHOEPKE, S.: Mobile XMPP and cloud service collaboration: An alliance for flexible disaster management. In: 2011 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011, S. 201–210

[KJPR12] KIM, ANBIN ; JEONG, SEONG-HO ; PARK, PYUNG-KOO ; RYU, HO YONG: QoS support for advanced multimedia systems. In: 2012 International Conference on Information Networking (ICOIN), 2012, S. 453–456

[KRES14] KHATTAK, HASAN ALI ; RUTA, MICHELE ; EUGENIO ; SCIASCIO, DI: CoAP-based healthcare sensor networks: A survey. In: : IEEE, 2014 — ISBN 978-1-4799-2319-9, S. 499–503

[KSHF11] KARNEGES, JUSTIN ; SAINT-ANDRE, PETER ; HILDEBRAND, JOE ; FORNO, FABIO ; CRIDLAND, DAVE ; WILD, MATTHEW: Stream Management. URL http://xmpp.org/extensions/xep-0198.html. - abgerufen am 2014-06-28. — This specification defines an XMPP protocol extension for active management of an XML stream between two XMPP entities, including features for stanza acknowledgements and stream resumption.

[KuAn11] KUMAR, S.P.S. ; ANAND, S.V.: A novel scalable software platform on android for efficient QoS on android mobile terminals based on multiple radio access technologies. In: Wireless Telecommunications Symposium (WTS), 2011, 2011, S. 1–6

[LBSM09] LUDWIG, SCOTT ; BEDA, JOE ; SAINT-ANDRE, PETER ; MCQUEEN, ROBERT ; EGAN, SEAN ; HILDEBRAND, JOE: Jingle. URL http://xmpp.org/extensions/xep-0166.html. - abgerufen am 2014-04-08. — This specification defines an XMPP protocol extension for initiating and managing peer-to-peer media sessions between two XMPP entities in a way that is interoperable with existing Internet standards. The protocol provides a pluggable model that enables the core session management semantics (compatible with SIP) to be used for a wide variety of application types (e.g., voice chat, video chat, file transfer) and with a wide variety of transport methods (e.g., TCP, UDP, ICE, application-specific transports).

[LKHJ13] LEE, SHINHO ; KIM, HYEONWOO ; HONG, DONG-KWEON ; JU, HONGTAEK: Correlation analysis of MQTT loss and delay according to QoS level.

In: : IEEE, 2013 — ISBN 978-1-4673-5742-5, 978-1-4673-5740-1, 978-1-4673-5741-8, S. 714–717

[LOGM10] LEVORATO, MARCO ; O'NEILL, DANIEL ; GOLDSMITH, ANDREA ; MITRA, URBASHI: Optimization of ARQ Protocols in Interference Networks with QoS Constraints (arXiv e-print Nr. 1009.3961), 2010

[LuSh11] LUO, HONGLI ; SHYU, MEI-LING: Quality of service provision in mobile multimedia - a survey. In: Human-centric Computing and Information Sciences Bd. 1 (2011), Nr. 1, S. 1–15

[MaKB07] MAHAMBRE, S.P. ; KUMAR, M. ; BELLUR, U.: A Taxonomy of QoS-Aware, Adaptive Event-Dissemination Middleware. In: IEEE Internet Computing Bd. 11 (2007), Nr. 4, S. 35–44

[Mhol00] MHOLDMANN: The choice of protocol for IoT and M2M will dictate the emergence and success of the market. URL http://mholdmann.wordpress.com/2013/05/17/the-choice-of-protocol-for-iot-and-m2m-will-dictate-the-emergence-and-success-of-the-market/. - abgerufen am 2014-06-19. — Michael Holdmann

[MoLG02] MOHAPATRA, PRASANT ; LI, JIAN ; GUI, CHAO: QoS in Mobile Ad Hoc Networks, 2002

[NoSa05] NOTTINGHAM, M. ; SAYRE, R.: The Atom Syndication Format, Request for Comments : IETF, 2005. — Published: RFC 4287 (Proposed Standard)Updated by RFC 5988

[Ongl02] ONG, L. AND J. YOAKUM: An Introduction to the Stream Control Transmission Protocol (SCTP). URL http://www.rfc-editor.org/rfc/rfc3286.txt. - abgerufen am 2014-05-20

[Pard03] PARDO-CASTELLOTE, G.: OMG Data-Distribution Service: architectural overview. In: 23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings, 2003, S. 200–206

[PaRR11] PARRA, O.J.S. ; RIOS, A.P. ; RUBIO, G.L.: Quality of Service over IPV6 and IPV4. In: 2011 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), 2011, S. 1–4

[PoNW07] PONGTHAWORNKAMOL, T. ; NAHRSTEDT, K. ; WANG, GUIJUN: The Analysis of Publish/Subscribe Systems over Mobile Wireless Ad Hoc Networks. In: Fourth Annual International Conference on Mobile and Ubiquitous Systems: Networking Services, 2007. MobiQuitous 2007, 2007, S. 1–8

[Post00a] POSTEL, J.: Transmission Control Protocol. URL http://tools.ietf.org/html/rfc793. - abgerufen am 2014-05-20

[Post00b] POSTEL, J.: User Datagram Protocol. URL http://tools.ietf.org/html/rfc768. - abgerufen am 2014-05-20

[Puza00] PUZANOV, OLEG: IoT Primer: IoT Protocol Wars: MQTT vs CoAP vs XMPP. URL http://www.iotprimer.com/2013/11/iot-protocol-wars-mqtt-vs-coap-vs-xmpp.html. - abgerufen am 2014-06-19

[RaAn12] RAM JEYARAMAN ; ANGUS TELFER: OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. URL http://www.amqp.org/specification/1.0/amqp-org-download. - abgerufen am 2014-04-09. — OASIS Standard

[RaRG14] RAPHAEL COHN ; RICHARD COPPEN ; GEOFF BROWN: OASIS Message Queuing Telemetry Transport (MQTT) TC. URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt. - abgerufen am 2014-04-09. — OASIS - Advancing open standards for the information society

[RoHo00] ROUHANA, N. ; HORLAIT, E.: Differentiated services and integrated services use of MPLS. In: Fifth IEEE Symposium on Computers and Communications, 2000. Proceedings. ISCC 2000, 2000, S. 194–199

[Sain11a] SAINT-ANDRE, P.: Extensible Messaging and Presence Protocol (XMPP): Core, Request for Comments : IETF, 2011. — Published: RFC 6120 (Proposed Standard)

[Sain11b] SAINT-ANDRE, P.: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, Request for Comments : IETF, 2011. — Published: RFC 6121 (Proposed Standard)

[Sain11c] SAINT-ANDRE, P.: Extensible Messaging and Presence Protocol (XMPP): Address Format, Request for Comments : IETF, 2011. — Published: RFC 6122 (Proposed Standard)

[Schn00] SCHNEIDER, STAN: Understanding The Protocols Behind The Internet Of Things | Embedded content from Electronic Design. URL http://electronicdesign.com/embedded/understanding-protocols-behind-internet-things. - abgerufen am 2014-01-03. — Electronic Design

[SGCI12] SODERMAN, P. ; GRINNEMO, K. ; CHEIMONIDIS, G. ; ISMAILOV, YURI ; BRUNSTROM, A.: An SCTP-based Mobility Management Framework for Smartphones and Tablets. In: 2012 26th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2012, S. 1107–1112

[ShJa11] SHUMINOSKI, T. ; JANEVSKI, T.: Adaptive QoS Cross-Layer module for multimedia services in heterogeneous wireless networks. In: 2011 10th International Conference on Telecommunication in Modern Satellite Cable and Broadcasting Services (TELSIKS). Bd. 2, 2011, S. 654–657

[Sina00] SINA GRUNAU: Protokolle zum Streaming von Eventdaten in Serviceumgebungen

[STKA13] SPAPIS, P. ; THEODOROPOULOS, P. ; KATSIKAS, G. ; ALONISTIOTI, N. ; GEORGOULAS, S.: A scheme for adaptive self-diagnosis of QoS degradation in future networks. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), 2013, S. 724–727

[Stpe00] STPETER: Protocols – The XMPP Standards Foundation. URL http://xmpp.org/xmpp-protocols/. - abgerufen am 2014-04-08

[Tark12] TARKOMA, SASU: Publish / Subscribe Systems : Design and Principles. 1. Aufl. Hoboken : Wiley, 2012 — ISBN 9781118354292

[Tbra14] T. BRAY, ED: The JavaScript Object Notation (JSON) Data Interchange Format (2014)

[TuRi00] TUEXEN, MICHAEL ; RIEGEL, MAXIMILIAN: Mobile SCTP. URL https://tools.ietf.org/html/draft-riegel-tuexen-mobile-sctp-09. - abgerufen am 2014-06-26

[VKBG95] VOGEL, A. ; KERHERVE, B. ; VON BOCHMANN, G. ; GECSEI, J.: Distributed multimedia and QOS: a survey. In: IEEE MultiMedia Bd. 2 (1995), Nr. 2, S. 10–19

[WiFr09] WILSON, STEPHEN ; FREY, JEREMY: The smartLab: Experimental and environmental control and monitoring of the chemistry laboratory. In: : IEEE, 2009 — ISBN 978-1-4244-4584-4, S. 85–90

[YoAT10] YOKOTA, K. ; ASAKA, T. ; TAKAHASHI, T.: QoS control mechanism based on flow rate to improve quality of short flows and low-rate flows. In: 2010 IEEE Network Operations and Management Symposium (NOMS), 2010, S. 272–277

[Zach00] <ZACH@SENSINODE.COM>, ZACH SHELBY: Constrained RESTful Environments (CoRE) Link Format. URL http://tools.ietf.org/html/rfc6690. - abgerufen am 2014-06-19

[ZhWG12] ZHENG, TAO ; WANG, LAN ; GU, DAQING: A Flow Label Based QoS Scheme for End-to-End Mobile Services. In: , 2012 — ISBN 978-1-61208-186-1, S. 169–174