

Institut für Mathematik und Informatik

Fernuniversität Hagen

Vergleichende Implementierung und Evaluierung einer ereignisgesteuerten, nicht blockierenden I/O Lösung für eine datenintensive Real-Time Webanwendung in Javascript und Dart

Bachelorarbeit

Barbara Drücke
Matrikel-Nummer 7397860

| | |
|--------------------|--------------------|
| Betreuer | Dr. Jörg Brunsmann |
| Erstprüfer | Prof. Hemmje |
| Zweitprüfer | Dr. Jörg Brunsmann |

Inhaltsverzeichnis

| | |
|-----------------------------------------------------------------------------|----|
| Abbildungsverzeichnis | IV |
| Tabellenverzeichnis | V |
| 1 Einleitung | 1 |
| 1.1 Motivation für diese Arbeit | 2 |
| 1.2 Aufbau der Arbeit | 3 |
| 2 Realtime-Schiffsverfolgung per AIS-Daten-Strom | 4 |
| 2.1 Anwendungsfälle | 4 |
| 2.2 Beschreibung der Anforderungen | 5 |
| 2.3 Grobentwurf der Anwendung | 6 |
| 3 Grundlagen | 8 |
| 3.1 Automatisches Informationssystem | 8 |
| 3.2 Websockets | 10 |
| 3.2.1 Bidirektionale Kommunikation im Web | 10 |
| 3.2.2 Node.js | 11 |
| 3.3 Google Dart | 12 |
| 3.3.1 Motivation für Dart | 12 |
| 3.3.2 Eigenschaften und Besonderheiten der Programmiersprache Dart . . | 12 |
| 3.3.3 Dart Tools (DartEditor, dart2js-compiler) | 13 |
| 3.3.4 Einbindung von Javascript-Bibliotheken in Dart mit js-interop | 13 |
| 4 Vorstellung der ausgeführten Implementierungen | 15 |
| 4.1 Implementierung des Prototypen in Javascript | 15 |
| 4.1.1 socket.io-Server | 16 |
| 4.1.2 socket.io-Client | 16 |
| 4.2 Vergleichsimplementierung in Google Dart | 16 |
| 4.2.1 HTML5-Server | 17 |

| | | |
|-------|---------------------------------------------------|----|
| 4.2.2 | js-Client | 17 |
| 4.2.3 | dart-Client | 17 |
| 5 | Vergleichende Evaluation | 19 |
| 5.1 | Socket.io-Websocket vs. HTML5-Websocket | 19 |
| 5.1.1 | Implementierungsaufwand | 19 |
| 5.1.2 | Latenzzeit | 19 |
| 5.1.3 | Performance | 21 |
| 5.1.4 | Browserunterstützung | 21 |
| 5.2 | Javascript-Client vs. Dart-Client | 21 |
| 5.2.1 | Implementierungsaufwand | 21 |
| 5.2.2 | Latenzzeit | 21 |
| 5.2.3 | Performance | 21 |
| 5.2.4 | Browserunterstützung | 23 |
| 6 | Fazit | 25 |
| 6.1 | Ergebnisse | 25 |
| 6.2 | Ausblick | 25 |
| | Literaturverzeichnis | 26 |

Abbildungsverzeichnis

| | | |
|-----|--------------------------------------------------------------------------------------------------------|----|
| 1.1 | Vesseltracker_Webapplikation | 1 |
| 1.2 | Cockpit_Elbe | 2 |
| 2.1 | Architektur-Entwurf der Realtime_Webapplikation | 6 |
| 5.1 | socket.io-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe | 20 |
| 5.2 | HTML5-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe | 20 |

Tabellenverzeichnis

| | | |
|-----|-----------------------------------------------------------------|----|
| 3.1 | Intervalle, in denen ein Schiff seine Daten aussendet | 9 |
| 3.2 | Die wichtigsten AIS-Telegrammtypen | 10 |
| 4.1 | Übersicht über Server-und Clientimplementierungen | 18 |

1 Einleitung

Die Vesseltracker.com GmbH ist ein Schiffsmonitoring und -reporting-Dienstleister. Der kostenpflichtige Dienst stellt den Kunden umfangreiche Informationen zu Schiffen weltweit zur Verfügung. Dabei handelt es sich einerseits um Schiffs-Stammdaten und andererseits um Schiffs-Positionsdaten. Die Positionsdaten sind AIS (Automatic Identification System) -Daten, wie sie von allen Schiffen über Funk regelmäßig zu senden sind.

Vesseltracker.com unterhält ein Netzwerk von ca. 800 terrestrischen AIS-Antennen, mit denen küstennahe AIS-Meldungen empfangen und via Internet an einen zentralen Rohdatenserver geschickt werden. Der Rohdatenserver verarbeitet die Meldungen und gibt sie umgewandelt und gefiltert an die Anwendungen des Unternehmens weiter. Zusätzlich erhält das Unternehmen AIS-Daten via Satellit über einen Kooperationspartner. Damit werden die küstenfernen Meeresgebiete und Gegenden, in denen Vesseltracker.com keine AIS-Antenne betreibt, abgedeckt. Die Kernanwendung des Unternehmens ist eine Webanwendung, die die terrestrischen AIS-Daten in einer Geodatenbank speichert und sie mit den Schiffs-Stammdaten und Satelliten-AIS-Daten in Beziehung setzt.

Für eine geographische Visualisierung der Schiffspositionen existiert das sogenannte 'Cockpit', wo die Schiffe als Icons auf Openstreetmap-Karten dargestellt werden. Diese

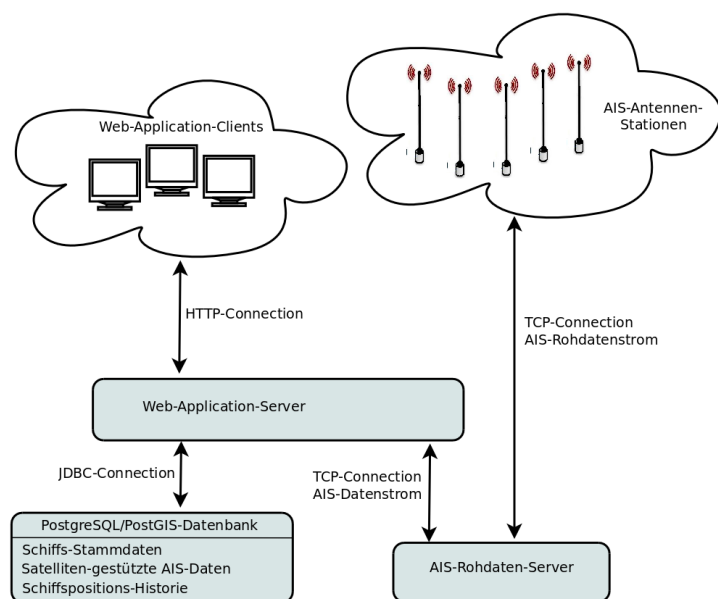


Abbildung 1.1: Vesseltracker_Webapplikation

1 Einleitung

Karte zeigt jeweils alle Schiffe an, die sich in dem frei wählbaren Kartenausschnitt zu der Zeit befinden. Aktualisiert werden die Positionsinformationen jeweils bei Änderung des betrachteten Bereichs oder einmal pro Minute. Detailinformationen erhält der Nutzer durch ein Click-Popup über das Icon des Schiffes. Darüber kann er sich auch die gefahrene Route der letzten 24 h anzeigen lassen.

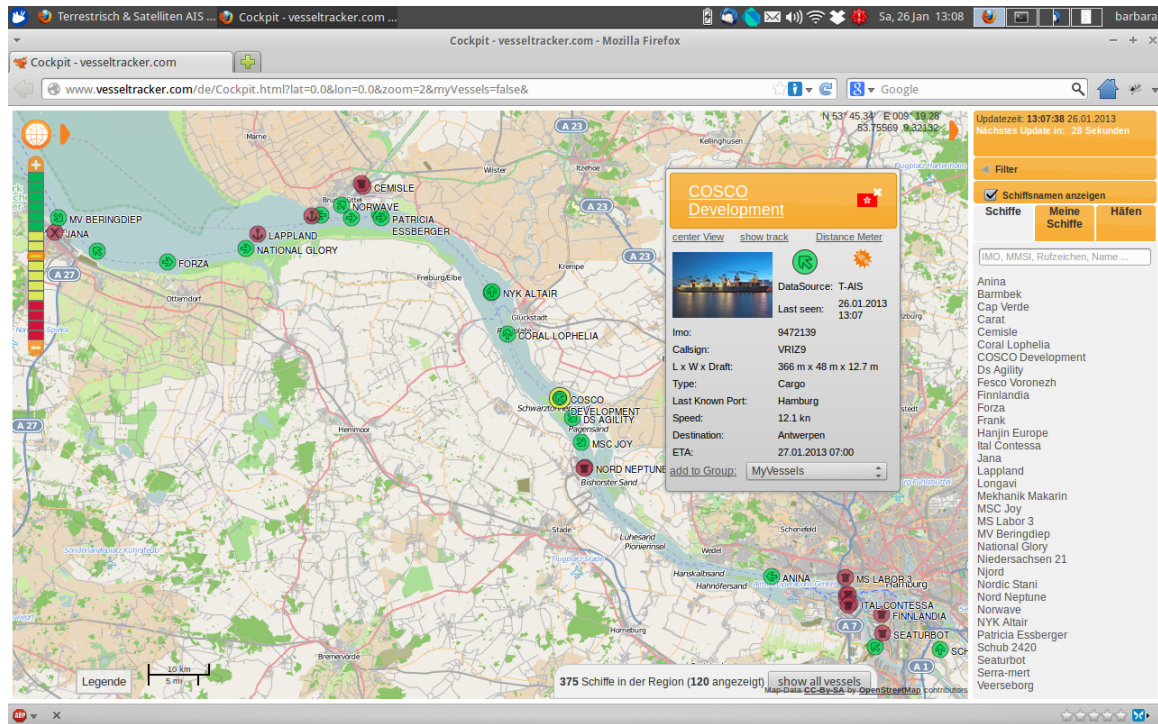


Abbildung 1.2: Cockpit_Elbe

1.1 Motivation für diese Arbeit

Aus mehreren Gründen erscheint es angebracht, die geographischen Schiffspeditionen nicht nur über die Cockpit-Anwendung anzubieten, sondern alternativ als real-time-Darstellung.

- Aufgrund der herausragenden Qualität des vesseltracker.com Antennen-Netzwerks sind die verfügbaren AIS-Daten aktuell, aktualisieren sich kontinuierlich und erreichen eine hohe weltweite Abdeckung. Damit ist es möglich, die Schiffsverkehrslage beliebiger Häfen, Wasserstraßen, Küstengebiete weltweit und sekundengenau zu präsentieren.
- Ein Phänomen in der menschlichen Wahrnehmung lässt die geplante Anwendung sehr viel zweckmäßiger erscheinen als die bisherige Cockpit-Anwendung. Aufgrund

der sogenannten Veränderungsblindheit oder “Change Blindness” werden Veränderungen an einem Objekt (in diesem Fall die Position eines Schiffs-Icons auf der Karte) in der Wahrnehmung überdeckt, wenn im selben Augenblick Veränderungen an der Gesamtsicht vonstatten gehen. Im Cockpit werden nach dem Laden neuer Positionsdaten alle Schiffsicons neu gerendert und unter Umständen Namens-Fähnchen gelöscht oder hinzugefügt, was zu einem kurzen “Flackern” führt. Dadurch ist es dem Betrachter nahezu unmöglich, die Positionsänderung eines Schiffes auf der Karte nachzuvollziehen.

- Real-time-Anwendungen gewinnen zunehmend an Bedeutung. Ihre Verbreitung wird durch den Fortschritt der verfügbaren Webtechnologien auf breiter Basis unterstützt. Mitbewerber auf dem Markt für AIS-Daten (z.B. Fleetmon.com) bieten bereits Echtzeit-Darstellungen ihrer AIS-Daten an. Um in diesem Geschäftsfeld weiterhin eine Spitzenposition innezuhaben, ist eine Realtime zwingend erforderlich, in einem nächsten Schritt sicher auch als Anwendung für Mobile Devices.

1.2 Aufbau der Arbeit

Im Kapitel 2 werden mögliche Anwendungs-Szenarien genauer beleuchtet und die funktionalen und nicht funktionalen Anforderungen an die geplante Anwendung herausgestellt. Anschließend wird die Systemarchitektur der geplanten Anwendung grob entworfen. Kapitel 3.1 gibt eine kurze Einführung in die Websocket-Technologie, Kapitel 3.2 stellt die Programmiersprache Google Dart vor. Kapitel 4 begründet die getroffene Auswahl an Websockets und schildert die Vorgehensweise bei der Implementierung, wobei Teile der Implementierung vorgestellt werden. Die ausgearbeiteten Implementierungen werden dann in Kapitel 5 nach verschiedenen Aspekten verglichen und die Ergebnisse in Kapitel 6 zusammengefasst.

2 Realtime-Schiffsverfolgung per AIS-Daten-Strom

2.1 Anwendungsfälle

Hafendienstleister wie Schlepper, Lotsen oder Festmacher verschaffen sich über einen Monitor einen Überblick über die Arbeitsvorgänge in ihrem jeweiligen Heimathafen, z.B. welche Schlepper welches Schiff schleppen, wo Lotsen an oder von Bord gehen, von welchen Tankern Schiffe betankt werden. Sie kontrollieren die eigenen Aufträge oder auch die der Mitbewerber. Die Anwendung läuft hierbei eher statisch, das heißt Zoomstufe und Kartenausschnitt ändern sich nur selten. Es ist also notwendig, dass die Anwendung unabhängig von Aktionen des Nutzers sich laufend oder regelmäßig aktualisiert.

Reedereien beobachten das Einlaufen, Anlegen, Festmachen, Ablegen und Auslaufen ihrer Schiffe in entfernten Häfen, wo es keine Unternehmensniederlassung gibt. Zum Beispiel kontrollieren sie, wann, an welchen Liegeplätzen ein Schiff wie lange festmacht. Dazu ist es zum einen notwendig, jederzeit auf eine geringe Zoomstufe heraus- und auf einen anderen Hafen wieder hineinzoomen zu können. Zum anderen soll die Anwendung Schnittstellen bieten, damit zusätzliche Informationen aus dem vesseltracker.com Datenpool (in diesem Fall Liegeplatzinformationen) von der Anwendung abgerufen werden können.

Weitere Anwendungsfälle sind Nutzer aus der Passagierschiffahrt, Schiffsfotografen oder Sicherheitsorgane (z.B. die Wasserschutzpolizei), bei denen die Beobachtung / Überwachung bestimmter Wasserverkehrswege oder Häfen von besonderem Interesse ist.

Die vesseltracker.com GmbH nutzt die Realime-Anwendung, um die vom Unternehmen angebotenen Daten zu präsentieren und zu bewerben. Dabei ist es wichtig, dass die Anwendung gesendete AIS-Signale im Schnitt in weniger als einer Sekunde auf dem Monitor als Position oder Positionsänderung darstellen kann und dass die Schiffsbewegungen fließend ohne das in der Einleitung beschriebene "Flackern" dargestellt werden. Damit kann

vesseltracker.com die höhere Genauigkeit und Aktualität der eigenen Daten gegenüber denen anderer Anbieter herausstellen.

Die Anwendungsfälle verdeutlichen noch einmal, dass der zusätzliche Nutzen der Realtimeanwendung gegenüber der Cockpitanwendung nicht ausschließlich im Informationsgehalt liegt. Die Daten im Cockpit sind ja ebenfalls im Minutenbereich aktuell. Der Vorteil liegt vielmehr in der Lebendigkeit der Darstellung. Bewegte Darstellungen binden stärker und für einen längeren Zeitraum die Aufmerksamkeit des Betrachters.

2.2 Beschreibung der Anforderungen

Die funktionalen Anforderungen sind:

- als Datenquelle sollen ausschließlich die vom Rohdatenserver als JSON-Datenstrom zur Verfügung gestellten AIS-Informationen dienen
- Schiffe sollen an ihrer aktuellen (realtime) Position auf einer Karte im Browser dargestellt werden
- Positionsänderungen einzelner Schiffe sollen ad hoc sichtbar gemacht werden
- die Schiffsbewegungen auf der Karten sollen nicht sprunghaft, sondern fließend erscheinen (Animation der Schiffsbewegungen in dem Zeitraum zwischen zwei Positionsmeldungen)
- die Karte soll in 16 Zoomstufen die Maßstäbe von 1:2000 bis 1: 200 Mio abdecken
- Schiffe sollen auf der Karte als Icons dargestellt werden, die den Navigationsstatus und gegebenenfalls den Kurs widerspiegeln
- bei hoher Auflösung und ausreichend statischen AIS-Informationen soll ein Schiff als Polygon in die Karte eingezeichnet werden.
- bei geringer Auflösung ist ein Überblick über die Verteilung der empfangenen Schiffe zu vermitteln
- Detail-Informationen zu jedem Schiff sollen als Popups über das Icon abrufbar sein

Nicht funktionale Anforderungen sind:

- die von den Antennen empfangenen AIS-Daten sind mit minimaler Verzögerung (< 500 msec) auf der Karte darzustellen

- die Anwendung sollte ca. 300 Verbindungen gleichzeitig erlauben und skalierbar sein
- als Clients der Anwendung sollten die gängigsten Browser unterstützt werden (IE, Chrome, Firefox, Safari, Opera)
- die Implementierungen werden auf Github als privates repository gehalten
- als Kartenmaterial sind die von vesseltracker gehosteten OpenstreetMap-Karten zu verwenden
- verwendete Software-Module sollten frei zugänglich sein (open source)
- ein Prototyp der Anwendung soll schnell zur Verfügung stehen. Dieser Prototyp soll Mitarbeitern und Partnern ermöglichen, ihre Anforderungen genauer zu spezifizieren oder sogar neue Anforderungen zu formulieren.

2.3 Grobentwurf der Anwendung

Die eingehende Schnittstelle der zu erstellenden Anwendung ist die Verbindung zum Rohdaten-server, die als TCP-Verbindung ausgeführt ist und einen JSON-Datenstrom liefert. Die ausgehende Schnittstelle ist der HTTP-Client (Browser). Zu erstellen ist also eine Client-Server-Anwendung, in der der Server zweifaches zu leisten hat, nämlich

1. eine tcp-socket-Verbindung zum Rohdatenserver zu unterhalten und
2. eine bidirektionale Verbindung zum HTTP-Client zu halten, in der der Client jederzeit Änderungen des betrachteten Kartenausschnittes an den Server senden und der Server jederzeit den Client über relevante, aus dem

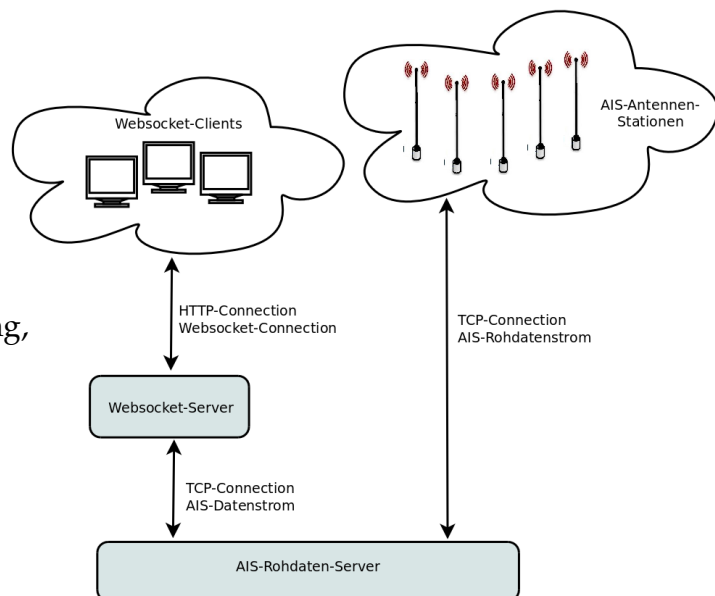


Abbildung 2.1: Architektur-Entwurf
Realtime_Webapplikation

der

2 Realtime-Schiffsverfolgung per AIS-Daten-Strom

JSON-Datenstrom ausgelesene, Schiffsbewegungen im betrachteten Kartenausschnitt informieren kann.

3 Grundlagen

3.1 Automatisches Informationssystem

Das Automatic Identification System (AIS) ist ein UKW-Funksystem im Schiffsverkehr, das seit 2004 für alle Berufsschiffe über 300 BRZ in internationaler Fahrt und seit 2008 auch für solche über 500 BRZ in nationaler Fahrt verpflichtend eingeführt worden ist. Es soll dabei helfen, Kollisionen zwischen Schiffen zu verhüten und die landseitige Überwachung und Lenkung des Schiffsverkehrs zu erleichtern. Außerdem verbessert AIS die Planung an Bord, weil nicht nur Position, Kurs und Geschwindigkeit der umgebenden Schiffe übertragen werden, sondern auch Schiffsdaten (Schiffsname, MMSI-Nummer, Funkrufzeichen, etc.). AIS ist mit UKW-Signalen unabhängig von optischer Sicht und Radarwellenausbreitung.

Für die Nutzung von AIS ist ein aktives, technisch funktionsfähiges Gerät an Bord Voraussetzung, das sowohl Daten empfängt als auch Daten sendet. Für Schiffe der Berufsschiffahrt sind Klasse-A-Transceiver an Bord vorgesehen, für nicht ausrüstungspflichtige Schiffe genügen Klasse-B-Transceiver, die mit niedriger VHF-Signalstärke und weniger häufig senden.

Die dynamischen Schiffsdaten (LAT, LON, COG, SOG, UTC) erhält der AIS-Transceiver vom integrierten GPS-Empfänger, bei Klasse A auch von der Navigationsanlage des Schiffes. Die Kursrichtung (Heading = HDG) kann über eine NMEA-183-Schnittstelle vom Kompass eingespeist werden.

Die AIS-Einheit sendet schiffsspezifische Daten, die von jedem AIS-Empfangsgerät in Reichweite empfangen und ausgewertet werden können: Statische Schiffsdaten:

- IMO-Nummer
- Schiffsname
- Rufzeichen
- MMSI-Nummer
- Schiffstyp (Frachter, Tanker, Schlepper, Passagierschiff, SAR, Sportboot u. a.)

3 Grundlagen

- Abmessungen des Schiffes (Abstand der GPS-Antenne von Bug, Heck, Backbord- und Steuerbordseite)

Dynamische Schiffsdaten

- Navigationsstatus (unter Maschine, unter Segeln, vor Anker, festgemacht, manövrierunfähig u. a.)
- Schiffsposition (LAT, LON, in WGS 84)
- Zeit der Schiffsposition (nur Sekunden)
- Kurs über Grund (COG)
- Geschwindigkeit über Grund (SOG)
- Vorausrichtung (HDG)
- Kursänderungsrate (ROT)

Reisedaten

- aktueller maximaler statischer Tiefgang in dm
- Gefahrgutklasse der Ladung (IMO)
- Reiseziel (UN/LOCODE)[5]
- geschätzte Ankunftszeit (ETA)
- Personen an Bord

Der Navigationsstatus und die Reisedaten müssen vom Wachoffizier manuell aktualisiert werden. Gesendet werden die AIS-Signale auf zwei UKW-Seefunkkanälen (Frequenzen 161,975 MHz und 162,025 MHz), wobei die Sendeintervalle abhängig sind von der Klasse, dem Manöverstatus und der Geschwindigkeit.

| Klasse | Manöver-Status | Geschwindigkeit | Sendeintervall |
|---------|------------------------|-----------------|----------------|
| Class A | geankert/festgemacht | <3kn | 3 min |
| Class A | geankert/festgemacht | >3kn | 10 sec |
| Class A | in Fahrt | 0-14kn | 10 sec |
| Class A | in Fahrt, Kursänderung | 0-14 | 3 1/3 sec |
| Class A | in Fahrt | 14-23kn | 6 sec |
| Class A | in Fahrt, Kursänderung | 14-23 | 2 sec |
| Class A | in Fahrt | >23kn | 2 sec |
| Class B | | <2 kn | 3 min |
| Class B | | >2 kn | 30 sec |

Tabelle 3.1: Intervalle, in denen ein Schiff seine Daten aussendet

Für AIS-Daten sind 22 standardisierte Nachrichtentypen bzw. Telegramme festgelegt: In dieser Arbeit werden nur Class A Positionsmeldungen betrachtet (Typ 1-3) un.

| ID | Nachrichtentyp |
|----|--------------------------------------------------------------------------|
| 1 | reguläre Positionsmeldung eines Klasse-A-Transceivers |
| 4 | Meldung einer Basisstation |
| 5 | reguläre Meldung von Schiffs- und Reisedaten eines Klasse-A-Transceivers |
| 9 | Positionsmeldung eines SAR-Luftfahrzeuges |
| 12 | sicherheitsbezogene Nachricht - adressiert |
| 14 | sicherheitsbezogene Nachricht - an alle |
| 18 | reguläre Positionsmeldung eines Klasse-B-Transceivers |
| 21 | Positions- und Statusmeldung eines AtoN-Transceivers |

Tabelle 3.2: Die wichtigsten AIS-Telegrammtypen

Zur landseitigen AIS-Infrastruktur gehören sogenannte AIS-Basisstationen und AIS-Empfänger. Basisstationen dienen einerseits zur Erfassung des Verkehrs in dem von ihnen abgedeckten Seegebiet, andererseits können diese Geräte die Übertragung von AIS-Transceivern an Bord gezielt steuern (z.B. Hochsetzen der Melderate). AIS-Empfänger sind reine AIS-Empfangsgeräte, die keine Daten senden.

3.2 Websockets

3.2.1 Bidirektionale Kommunikation im Web

In der Entwicklung der Kommunikationstechnologien im Internet galt lange Zeit das request/response Paradigma, nach dem Anfragen vom Client vom Server beantwortet werden. Dieses Paradigma wird Stück für Stück aufgebrochen durch kontinuierliche Weiterentwicklungen in Richtung einer bidirektionalen Kommunikation zwischen Server und Client. Mit HTTP Long Polling, HTTP Streaming und Ajax on demand ist es, nachdem der Client eine Verbindung hergestellt hat, für den Server möglich, beim Eintreffen neuer Daten scheinbar selbständig einen Datenaustausch zum Client zu initiieren. Dabei handelt es sich eigentlich nur um einen aufgeschobenen response auf einen zuvor gestellten client-Request. Der Nachteil dieser Technologien liegt darin, dass sie, weil sie http-Nachrichten austauschen, einen großen Überhang an Header-Informationen mitzusenden gezwungen sind, der sich in Summe negativ auf die Latenzzeit auswirkt. Damit sind diese Technologien für zeitkritische (realtime) Anwendungen nicht unbedingt geeignet. Das 2011 eingeführte WebSocket-Protokoll dagegen beschreibt eine API, die eine echte

bidirektionale Socket-Verbindung zwischen Server und Client ermöglicht, in der beide Seiten jederzeit Daten schicken können. Dieser Socket wird im Anschluss an einen initialen HTTP-handshake aufgebaut, indem Server und Client einen Upgrade der Verbindung auf das WebSocket-Protokoll aushandeln.

3.2.2 HTML5-WebSocket API-Spezifikation

3.3 Node.js

Node.js ist eine javascript-Bibliothek für performante, skalierbare und echtzeitfähige Serveranwendungen in Netzwerken. Diese Eigenschaften sind größtenteils dem Konzept des asynchronen, nicht blockierenden I/O von javascript im Allgemeinen und node.js im Besonderen geschuldet. Javascript ist per se asynchron konzipiert für die Verwendung im Webbrowser, wo synchrone Verarbeitung wegen der Verzögerung des Seitendarstellung nicht in Frage kommt. Den gleichen Ansatz übernimmt node.js für die Serverseite. Node.js arbeitet single-threaded und eventbasiert. Die zentrale Kontrollstruktur, die den Programmablauf steuert, ist der Event-Loop. Er empfängt Events, die von Programm- oder Nutzeraktionen ausgelöst werden und setzt sie in Callback-Funktionen um. Kommt es im Programmablauf zur Interaktion mit einer externen Ressource, wird diese Interaktion in einen neuen Prozess ausgelagert und mit einer Callback-Methode versehen. Anschließend kann der Event Loop weitere aufgelaufene Events verarbeiten. Ist die Interaktion abgeschlossen bekommt der Event Loop ein Signal und setzt beizeiten die Verarbeitung mit der entsprechenden Callback-Methode fort. Node.js bringt als Laufzeitumgebung die V8-Javascript-Engine mit, die die Ausführung von javascript-code durch Just-In-Time-Kompilierung optimiert. Außerdem bietet node.js eine direkte Unterstützung für das HTTP-Protokoll und das WebSocket-Protokoll. Zusammen mit der Unterstützung des JSON-Datenformats sind alle notwendigen Bausteine zusammen für skalierbare, echtzeitfähige Webanwendungen.

3.3.1 Nodejs-Websockets

Als konkrete Pakete für Websockets standen innerhalb von node.js zum Zeitpunkt der Implementierung (November 2012) die Bibliotheken websocket (<https://github.com/Worlize/WebSocket-Node>) und socket.io (<http://socket.io>) zur Verfügung. Die Bibliothek websocket genügt der HTML5-WebSocket-API-Spezifikation (s.o). Socket.io erweitert die Funktionalität um

eine Abstrahierung von verschiedenen Browsern, indem es auf andere Protokolle zurückgreift bei Browsern, die Websockets nicht unterstützen.

3.4 Google Dart

3.4.1 Motivation für Dart

Dart ist eine von der Firma Google als OpenSource Projekt seit ca. 2 Jahren explizit für Webanwendungen entwickelte Programmiersprache. Das Ziel ist es, eine Sprache zu entwickeln, die komplexe Webanwendungen besser unterstützt als Javascript mit seinen historisch bedingten Ungereimtheiten und Schwächen. Dart arbeitet ereignisbasiert, asynchron und in einem einzigen Thread. Dart läuft nativ in der Dart-Virtual-machine, kann aber auch nach Javascript kompiliert werden.

Das Entwicklerteam definiert die Design-Ziele folgendermaßen: Dart soll

- eine sowohl strukturierte als auch flexible Web-Programmiersprache sein
- sich für Programmierer vertraut anfühlen und intuitiv erlernbar sein
- mit seinen Sprachkonstrukten performant sein und schnell zur Ausführung kommen
- auf allen Webdevices wie Mobiles, Tablets, Laptops und Servern gleichermaßen lauffähig sein
- alle gängigen Browser unterstützen.

3.4.2 Eigenschaften und Besonderheiten der Programmiersprache Dart

Klassen

Klassen sind ein wohlbekanntes Sprachkonzept das zur Kapselung und Wiederverwendung von Methoden und Daten. Jede Klassen definiert implizit ein Interface.

Optionale Typisierung

Die Typisierung in Dart ist optional, das heißt, sie führt nicht zu Laufzeitfehlern, sondern ist lediglich als Werkzeug für den Entwickler gedacht zur besseren Verständlichkeit und als Hilfe beim Debuggen.

Gültigkeitsbereiche

Die Gültigkeitsbereiche von Variablen in Dart gehorchen einfachen intuitiv nachvollziehbaren Regeln: Variablen sind gültig in dem Block (...), in dem sie definiert sind.

Bibliotheken

Entwickler können Bibliotheken bauen und nutzen und sich darauf verlassen, dass diese nicht zur Laufzeit geändert werden -> shared libraries.

Isolates

Modern web browsers, even on mobile platforms, run on multi-core CPUs. To take advantage of all those cores, developers traditionally use shared-memory threads running concurrently. However, shared-state concurrency is error prone and can lead to complicated code. Instead of threads, all Dart code runs inside of isolates. Each isolate has its own memory heap, ensuring that no isolate's state is accessible from any other isolate.

Toolability

Dart is designed to work well with tools. A rich set of execution environments, libraries, and development tools have already been built to support the language. Dart's toolability enables productive and dynamic development, including edit-and-continue debugging and beyond—up to a style where you program an application outline, run it, and fill in the blanks as you run.

3.4.3 Dart Tools (DartEditor, dart2js-compiler)

3.4.4 Einbindung von Javascript-Bibliotheken in Dart mit js-interop

js-interop ist eine Dart Bibliothek, die es Dart-Anwendungen ermöglicht, javascript-Bibliotheken zu verwenden und zwar sowohl in nativem Dart code, der in der Dart-Virtual-Machine ausgeführt wird als auch in mit dart2js zu javascript kompilierten Dart-Code. Nachdem die Bibliothek in die Anwendung worden ist, kann ein sogenannter Proxy zum

javascript-kontext der Seite erstellt werden. Referenzen an diesen Proxy werden automatisch zu javascript umgeleitet. Auf oberster Ebene lassen sich damit javascript-maps und -arrays generieren, die mit den entsprechenden Objekten in Dart korrespondieren. Über diesen Proxy können auch Proxies zu beliebigen javascript-Objekten generiert werden, deren Eigenschaften und Methoden im javascript-Scope zur Verfügung stehen. Um Dart-Funktionen aus dem javascript-Scope heraus aufzurufen, wird die entsprechende Funktion in ein Callback-Objekt umgewandelt, das entweder ein einziges Mal oder mehrmals aufrufbar ist. Um die Lebensdauer dieser Proxies und Callback-Objekte zu verwalten benutzt Dart das Scope-Konzept. Per default haben alle proxies nur lokale Gültigkeit. Sollen sie den Ausführungszeitraum des scopes überdauern, können sie ausdrücklich aufbewahrt werden, müssen dann aber zu Vermeidung von memory leaks auch explizit wieder freigegeben werden. Dasselbe gilt für Callback-Objekte, die mehrmals aufrufbar sind.

Dart-Websockets mit Dart:IO

4 Vorstellung der ausgeführten Implementierungen

Aus den betrachteten Programmiersprachen (Javascript, Google Dart) und Modulen (node.js, socket.io, dart.io, dart.html) gilt es nun, sinnvolle und vergleichbare Lösungen zu implementieren für die Serveranwendung und die korrespondierenden Clients.

4.1 Implementierung des Prototypen in Javascript

Um der Anforderung nachzukommen, schnell einen funktionierenden Prototypen zu liefern, wird für die erste Implementierung Javascript als Programmiersprache gewählt. Damit ist eine Unterstützung der gängigen Browserclients gewährleistet und eine große Anzahl an Modulen steht zur Verfügung.

Node.js wird als Framework eingebunden, weil node.js

- ereignisgesteuerte Verarbeitung unterstützt, was für die Umsetzung der geforderten Funktionalität hilfreich ist,
- Asynchronizität von I/O-Operationen bietet, womit die Verarbeitungszeit minimal gehalten werden kann,
- mehrere Websocket-Implementierungen anbietet (socket.io, websocket, ws).

Für den Websocket wird das socket.io - package genutzt, weil socket.io neben zahlreichen Konfigurationsmöglichkeiten eine breite Browserunterstützung bietet, indem es mit Browsern, die Websockets nicht unterstützen, eine Verbindung auf Basis älterer Technologien wie http-Polling aufbaut. Dadurch erhöht sich für diese Clients zwar die Latenzzeit, die Funktionalität ist aber gewährleistet.

Im Folgenden wird die hier beschriebene Implementierung socket.io-Server genannt. Der zugehörige Client (socket.io-Client) wird ebenfalls in Javascript implementiert und bindet die socket.io-Bibliotheken ein.

4.1.1 socket.io-Server

Die Serveranwendung hat im Grunde zwei Aufgaben: erstens eine JSON-over-TCP-Verbindung zum Empfang der Daten vom Rohdatenserver und zweitens einen Websocket-Server zur Verteilung der Daten an die Clients. Weil Node.js singlethreaded ist (3.2.2) würden beide Aufgaben in einem einzigen Prozess bearbeitet. Um das Potential an Parallelverarbeitung eines Dualcore oder Multicore-Servers zu nutzen, ist es daher sinnvoll, mindestens zwei Prozesse zu generieren. Dazu wurde das node.js-Modul `child_process` genutzt. Die ausführbare Datei `master.js` generiert damit zuerst einen Prozess, der den AIS-Client (`ais_client.js`) startet und anschließend einen Prozess (`worker.js`), der einen Websocket-Server zur Verfügung stellt.

Der Datenaustausch zwischen beiden Prozessen funktioniert über zwei nosql-Datenbanken. In einer mongo-Datenbank werden alle vom `ais_client`-Prozess empfangen Positions- und Reisedaten unter der `mmsi` eines Schiffes gespeichert. Dabei wird die `upsert`-Option von Mongo genutzt, so dass nicht vorhandene Schiffe eingefügt und vorhandene aktualisiert werden.

Der `worker`-Prozess greift auf die Mongo-Datenbank zu, um für einen vom Client angefragten Kartenausschnitt die entsprechenden Schiffe abzufragen. Dabei wird der in Mongo zur Verfügung stehende Geo-Index auf der Schiffposition verwendet.

Zur Verteilung der Positionsmeldungen (`msgid` 1,2 und3) wird außerdem eine Redis-Datenbank verwendet, die über einen `publish/subscribe`-Mechanismus verfügt. Über einen Kanal "`vesselpos`" publiziert der AIS-Client-Prozess die Positionsmeldungen und der `worker`-Prozess meldet sich am selben Kanal an und wird über jede Positionsmeldung benachrichtigt, die er an seine verbundenen Websocket-Clients weiterreichen kann.

4.1.2 socket.io-Client

Das `socket.io` Paket bietet Features wie die interne Clientverwaltung durch den Websocket.

4.2 Vergleichsimpementierung in Google Dart

Nachdem die Anwendung als Prototyp in Javascript fertiggestellt ist, soll eine vergleichbare Implementierung in Google Dart realisiert werden. Das paket `dart:io` bietet die

entsprechende Unterstützung für HTML5-Websocket-Server und `dart:html` für HTML5-Websocket-Clients. Allerdings ergeben sich auf der Serverseite einige schwerwiegende Probleme: - zum Zeitpunkt der Umsetzung (Dezember 2012) fehlt noch ein Redis-Client in Dart, so dass für die `publish/subscribe`-Lösung mit Redis [siehe] eine Alternative entwickelt werden müsste, die wiederum die Vergleichbarkeit beider Implementierungen herabsetzt. - der `socket.io`-Server nutzt 'JSON over TCP', um die Daten vom Rohdaten-server abzufragen. 'JSON over TCP' ist in Dart (noch) nicht implementiert. Ohne die Schnittstelle zum Rohdatenserver zu verändern ist also keine vergleichbare Lösung in Dart umsetzbar.

Der Vergleich zwischen der Javascript- und der Google Dart-Anwendung ist also zu diesem Zeitpunkt lediglich auf der Clientseite möglich beziehungsweise sinnvoll. Websocketverbindungen werden in Dart clientseitig mit dem Paket `dart:html` unterstützt. Dabei handelt es sich allerdings um HTML5-Websockets [siehe Grundlagen]. Eine Unterstützung für `socket.io`-Websockets existiert in Dart noch nicht. Folglich muss neben dem `socket.io`-Server ein zweiter Server (in Javascript) implementiert werden, der eine Websocket-Verbindung nach der HTML5-Websocket-API-Spezifikation aufbaut. Dies ist relativ einfach möglich: in `node.js` kann hierfür das Modul `websocket` eingebunden werden. Im Folgenden wird dieser Server HTML5-Server genannt.

4.2.1 HTML5-Server





Für den HTML5-Server ist es nun möglich, zwei vergleichbare HTML5-Websocket-Clientanwendungen jeweils in Javascript (`js-client`) und Dart (`dart-client`) zu bauen, die beide eine Websocketverbindung nach der HTML5-Spezifikation zum HTML5-Server aufbauen.

4.2.2 js-Client

Die Funktionalität entspricht exakt der des `socket.io`-Clients.

4.2.3 dart-Client

nicht unterstützt Zum Schluß wird der Client für den HTML5-Server in Dart geschrieben.

| | | HTTP-Client | | |
|-------------|-----------------------------------|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| | | Javascript | | Google Dart |
| | | socket.io | HTML-5 | dart.html |
| HTTP-Server | HTML-5 websocket- Server | ✗ |   |  |
| | socket.io websocket- Server |  | ✗ | ✗ |



Server-Vergleichstest



Client-Vergleichstest

Tabelle 4.1: Übersicht über Server-und Clientimplementierungen

5 Vergleichende Evaluation

Die realisierten Implementierungen lassen zwei Vergleiche zu:

- Node.js-server mit socket.io-Websocket-Server vs. node.js-Server mit HTML5-Websocket-Server, wobei die Javascript-Clients sich nur marginal unterscheiden.
- Javascript-Client vs. Dart-Client, wobei beide auf denselben node.js-Server mit HTML5-Websocket-Server zugreifen

5.1 Socket.io-Websocket vs. HTML5-Websocket

5.1.1 Implementierungsaufwand

Anzahl zeilen code

5.1.2 Latenzzeit

querytime

time received

5 Vergleichende Evaluation

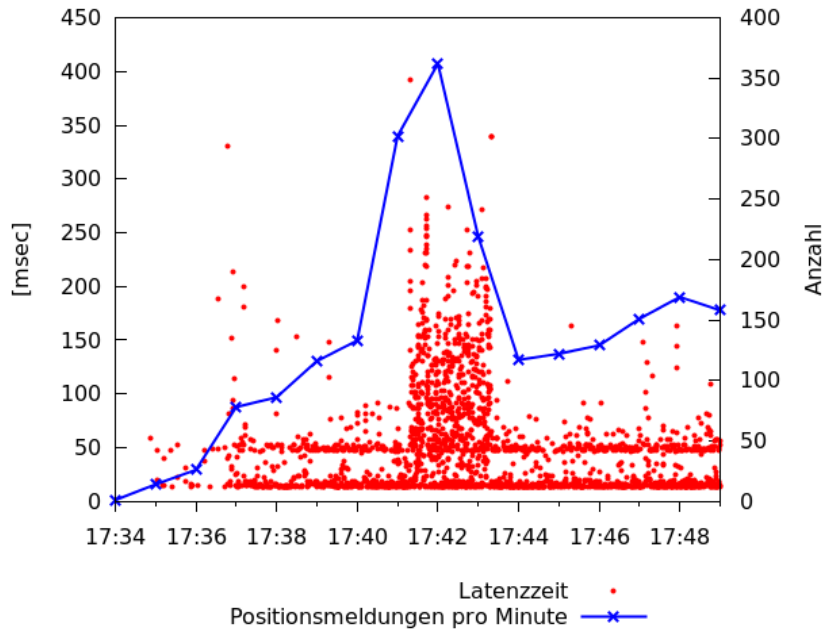


Abbildung 5.1: socket.io-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe

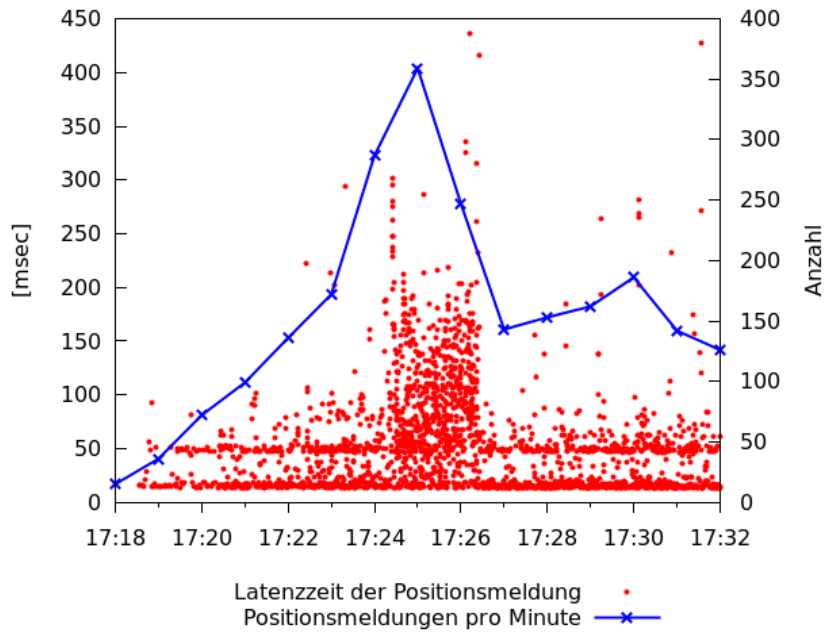


Abbildung 5.2: HTML5-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe

5.1.3 Performance

paintToMap

5.1.4 Browserunterstützung

Firefox, Chrome, IE, Safari

5.2 Javascript-Client vs. Dart-Client

5.2.1 Implementierungsaufwand

js-Client

Zeilen Code

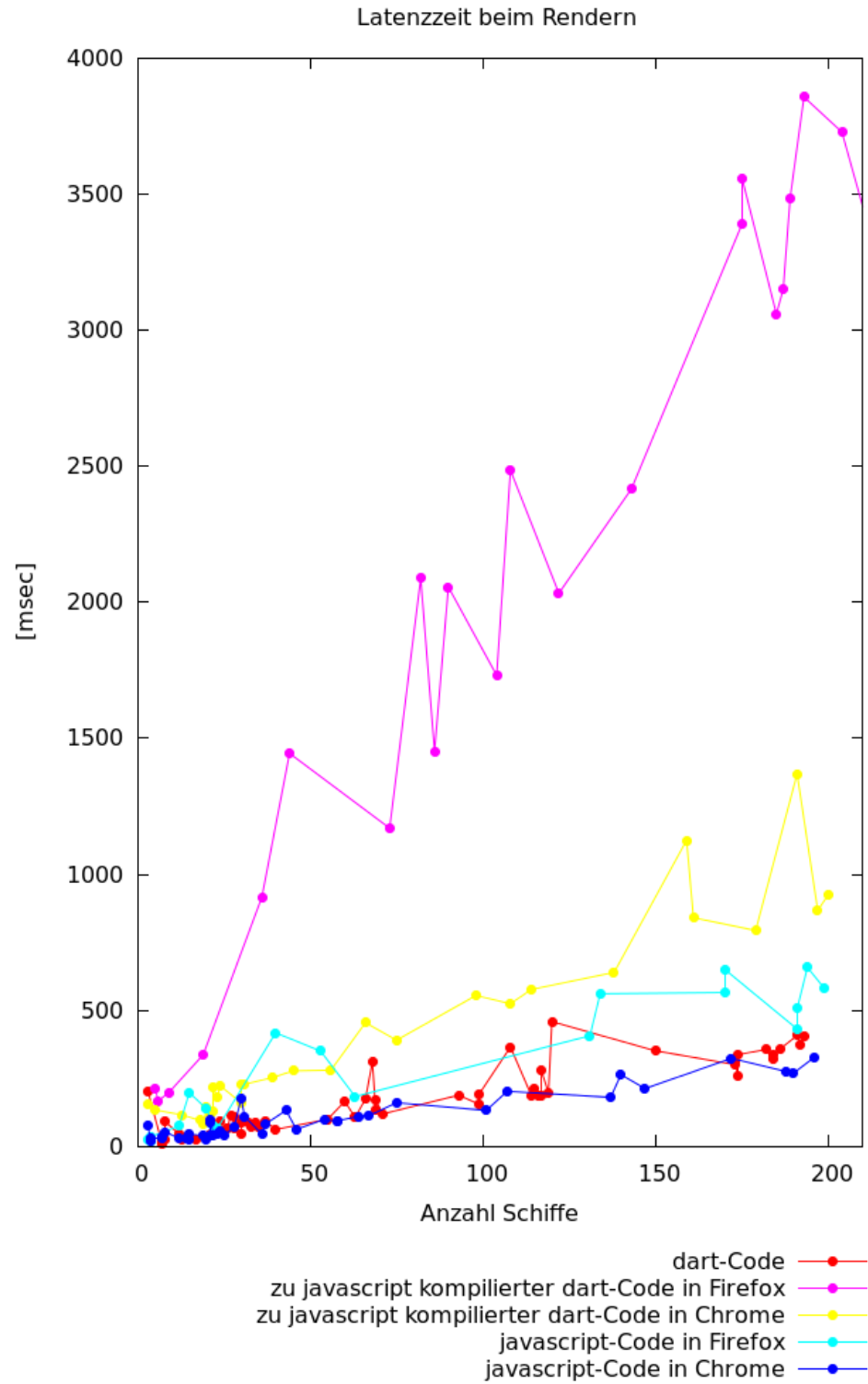
5.2.2 Latenzzeit

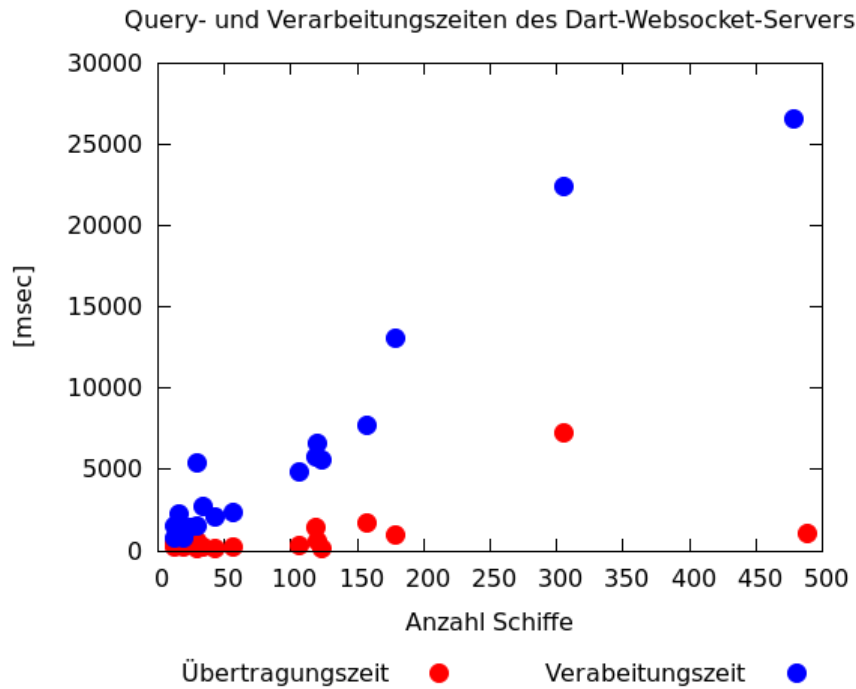
queryTime

5.2.3 Performance

paintToMap

5 Vergleichende Evaluation





5.2.4 Browserunterstützung

Dartium

Firefox, Chrome, IE, Safari

Der dart-Client kompiliert den in Dart geschriebenen Code zu Javascript.

Dabei traten Fehler auf, die unter Dartium (also im originalen Dart-Code) nicht auftraten. 1. Wird innerhalb des Javascript-Scopes eine Methode auf einen javascript-Proxy (hier `_map`) aufgerufen und ein proxy wird zurückgegeben, dann ist es nicht möglich auf diesen Proxy, der in diesem Fall vom Typ `LatLngBounds` sein müsste, eine Methode der Klasse `LatLngBounds` aufzurufen. => `TypeError: t1.get$_map(...).getBounds$0(...).getSouthWest$0 is not a function`

dart-client: web/leaflet_maps.dart

```
List getBounds() var south, west, north, east; js.scoped(() south= _map.getBounds().getSouthWest().lng;
west = _map.getBounds().getSouthWest().lat; north = _map.getBounds().getNorthEast().lng;
east = _map.getBounds().getNorthEast().lat; ); return [west, south, east, north];
```

In diesem Fall wird einfach als work-Around eine andere Methode verwendet (getBBoxString), die einen String mit den Bounds zurückgibt. Aus den Teilen dieses Strings werden mit der Methode parse(string) der Klasse double die Werte der Eckpunkte der Bounds generiert.

```
String getBounds() String bBox; js.scoped(() bBox = _map.getBounds().toBBoxString(); );  
return bBox;
```

Weil dadurch der message-Parameter 'bounds' kein number-Array, sondern ein String ist, muss im html5-Server der String einmal zum Float geparkt werden.

2. Ein Feld (IMO") wird auf null und auf > 0 geprüft.

6 Fazit

6.1 Ergebnisse

6.2 Ausblick

-Satellitendaten in die Anwendung einbinden

Literaturverzeichnis

- [1] Seth Ladd, SSorry, at the time of this writing, I'm not aware of a socket.io port for Dart. socket.io is nice because it has a bunch of implementation options for browsers that don't support Web sockets. Sounds like a good idea for a hackathon project!",2012 Oct 15, <http://stackoverflow.com/questions/12882112/is-there-a-socket-io-port-to-dart>.
- [2]
- [3] <http://nbn-resolving.de/urn:nbn:de:swb:14-1114955960020-08344>

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum

Unterschrift