Evaluation der ARVIDA-Referenzarchitektur

Methoden und Ergebnisse

Dr. Matthias Bues, Fraunhofer IAO, 26.10.2016, Kiel

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
DLR Projektträger

Evaluation der ARVIDA-Architektur

Ziele

- Begleitung und Steuerung der iterativen Entwicklung der Referenzarchitektur
- Sicherstellung der Kompatibilität und Integrierbarkeit von Technologien über Dienste
- Evaluation der Funktionalität und des Leistungsvermögens der Architektur
- Evaluation des Architekturkonzeptes
- Beitrag zur Entwicklung von Optimierungswerkzeugen

Funktionale vs. nicht-funktionale Anforderungen nach ISO 25010

Funktionale Anforderungen: Was soll das System leisten

- ARVIDA-Architektur = Rahmenwerk zur Verteilung
  - Korrekte Datenauslieferung
- ARVIDA Architektur: ARVIDA Komponenten: Nicht Bestandteil des Architekturrahmens muss spezifisch in den TP mit den UseCases evaluiert werden

Nichtfunktionale Anforderungen = In welcher Qualität soll das System die Leistung erbringen

- Interoperabilität -> Konformität mit den Vokabularen
- Flexibilität -> Austauschbarkeit von Komponenten
- Performanz
  - Skalierbarkeit
  - Durchsatz
  - Antwortzeitverhalten
- Nutzbarkeit für Entwickler

Begriffsdefinitionen

Verifikation vs Validierung (Boehm)

- Verifikation = Nachweis der Korrektheit („Doing things right“)
- Validierung = Nachweis der Tauglichkeit („Doing the right things“) ARVIDA-Evaluation

Was wird validiert?

- Erreichung der (nicht-formalen) Entwicklungsziele

Anforderungen an die Architektur definieren die Entwicklungsziele

- Interoperabilität
- Effizienz: Laufzeit, Ressourcen
- Usabillity/Nutzbarkeit für Entwickler

Validierungsmethoden

- Usability/Nutzbarkeit: Checklisten
- Interoperabilität und Effizienz: Tests

Anforderungserfüllung: Fullfillment by Design

Funktionalität

- Datenverteilung: Zentraler Punkt von REST Architekturen, Implementierungen vorhanden
- Skalierbarkeit: Zentraler Punkt von REST Architekturen, Implementierungen vorhanden
- Interoperabilität: Teilweise erfüllt durch Protokollkonformität (http)
- Flexibilität = Skalierbarkeit + Interoperabilität

Zugänglichkeit

Nutzung verbreiterer (Web-)Standards

- Offen zugänglich
- Viele Use-Cases
- Offene Referenzimplementierungen von Komponenten

Anforderungserfüllung: Zu validieren (nur nichtfunktionale Aspekte im Sinne der ISO 25010)

Interoperabilität

- Vokabulare = formalisierte Anforderungen
- Vokabularkonformität
- Diensteklassen mit garantierter Semantik
  - wichtig u.a.: sinnvolle Defaults

Performanz

- Wie schnell ? -> Zeit
- Wieviel ? -> Daten
- Wieviel gleichzeitig? -> Skalierbarkeit

Usability/Nutzbarkeit aus Entwicklersicht

- Wie schnell kann ich entwickeln?
  - Initial
  - Weiterentwicklung / Wartung
- Wie gut passt mein mentales Modell?
- Dokumentation, Werkzeuge, Beispiele

Korrektheit und Qualität

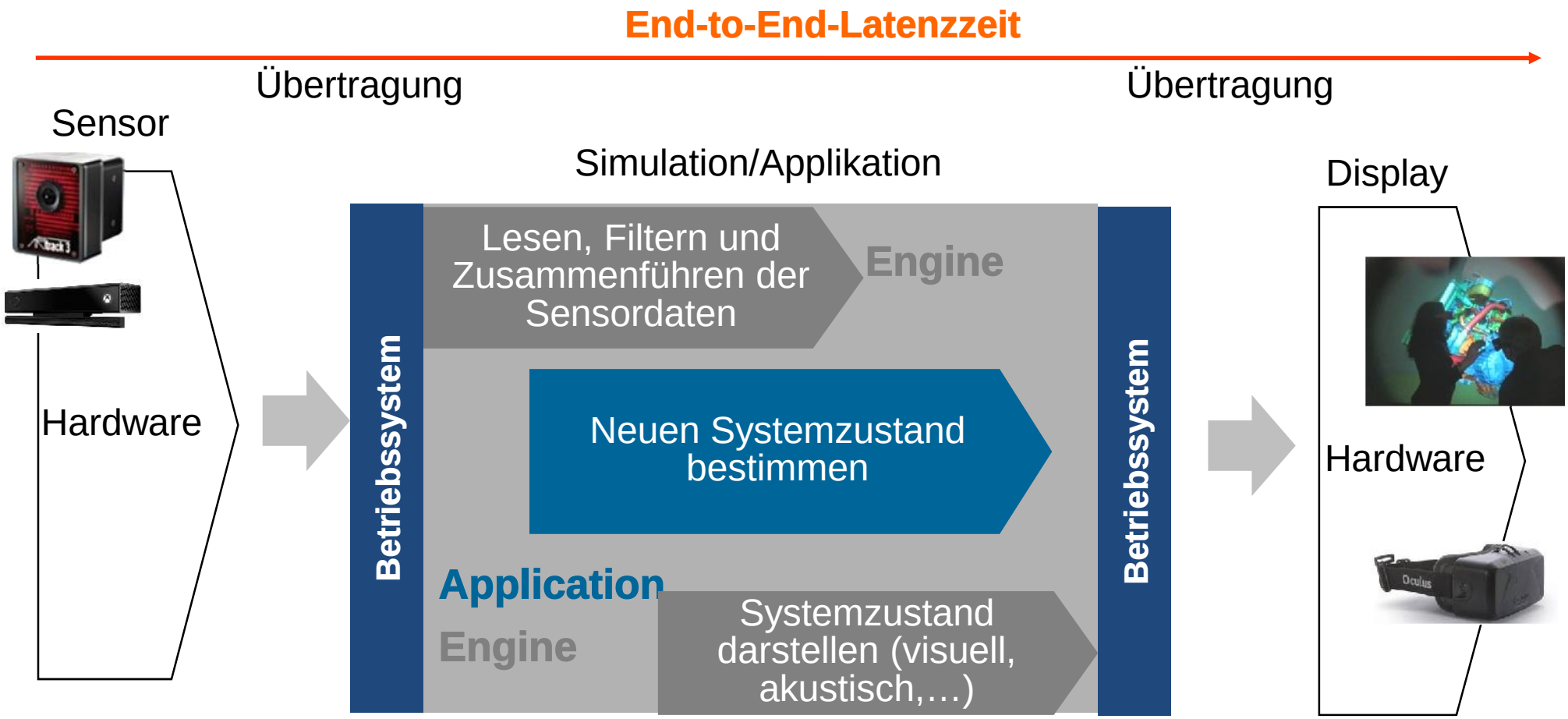
- Wie ist die Güte der Daten?
- Liefern Dienste (gleicher Klasse) die gleichen Daten?
- Wo kann man dies erwarten?

Interoperabilität: Vokabularkonformität

- Vokabularvollständigkeit
  - Manuelle Prüfung (= Lesen der Vokabulare) anhand der Use Cases
  - Iterativer Prozess

- Automatisiertes Testen gegen API -> Vollständigkeit im Sinne der Vokabulare
  - Ist Attribut vorhanden ?
  - Werden Werte gemäß der API geliefert ?
  - Defaults vorhanden ?
  - Nicht getestet: Semantik und Güte der Werte (hängen vom Dienst ab)

Interaktive Echtzeitsysteme: Kritischer Faktor Latenzzeit



vgl. Mine 1995, Steed 2008, Ellis 2000

Performanzmessungen

Dimensionen

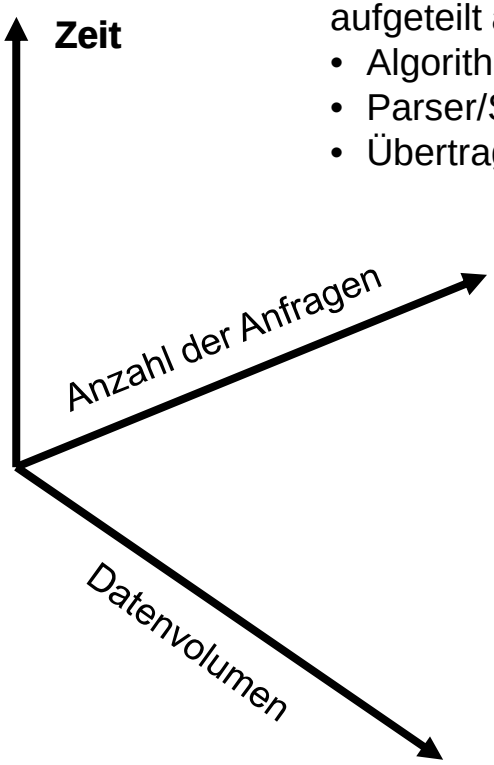
- Zeit: Messung
- Volumen: Vorgabe
- Anzahl Clients: Vorgabe

Abgeleitete Performanzmaße

- Latenz (Antwortzeit/Updatezeit)
- Jitter (Schwankungen der Antwortzeit)
- Volumen pro Zeiteinheit
- Zeit-Overhead der Architekturkomponenten

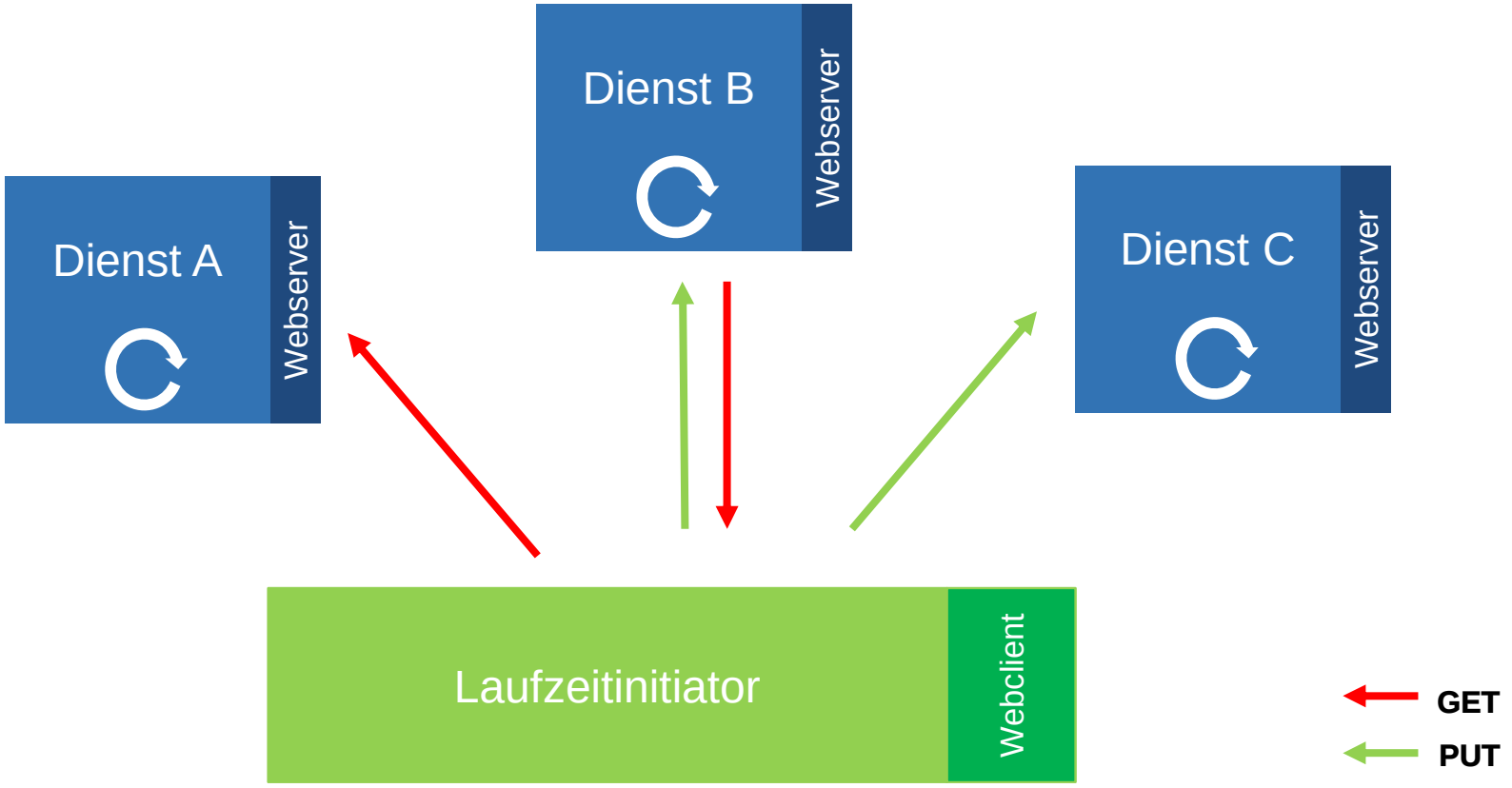
  $t_{Architektur} = t_{Parser} + t_{Serialisierung} + t_{Envelope}$

- Volumen-Overhead
  - Verhältnis der Datenmengen von Envelope/Header und Nutzdaten



aufgeteilt auf Komponenten
- Algorithmus
- Parser/Serialisierung/Envelope
- Übertragung

ARVIDA-Komponentensicht

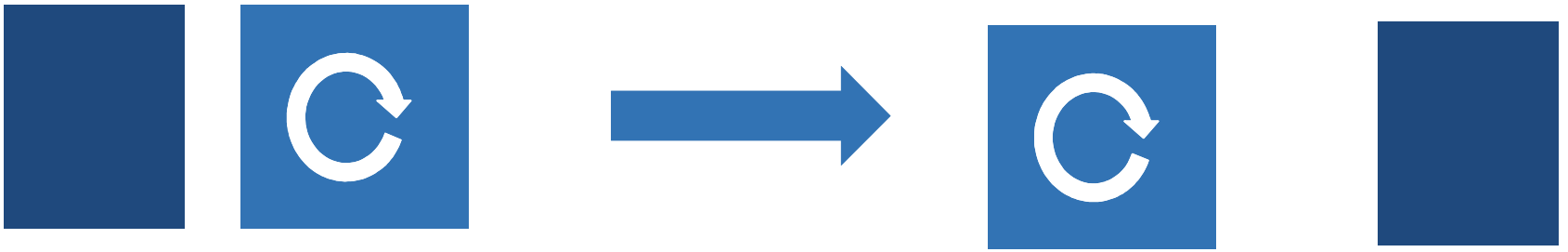


Zusammensetzung der End-to-End--Latenzzeit

End-to-End-Latenzzeit

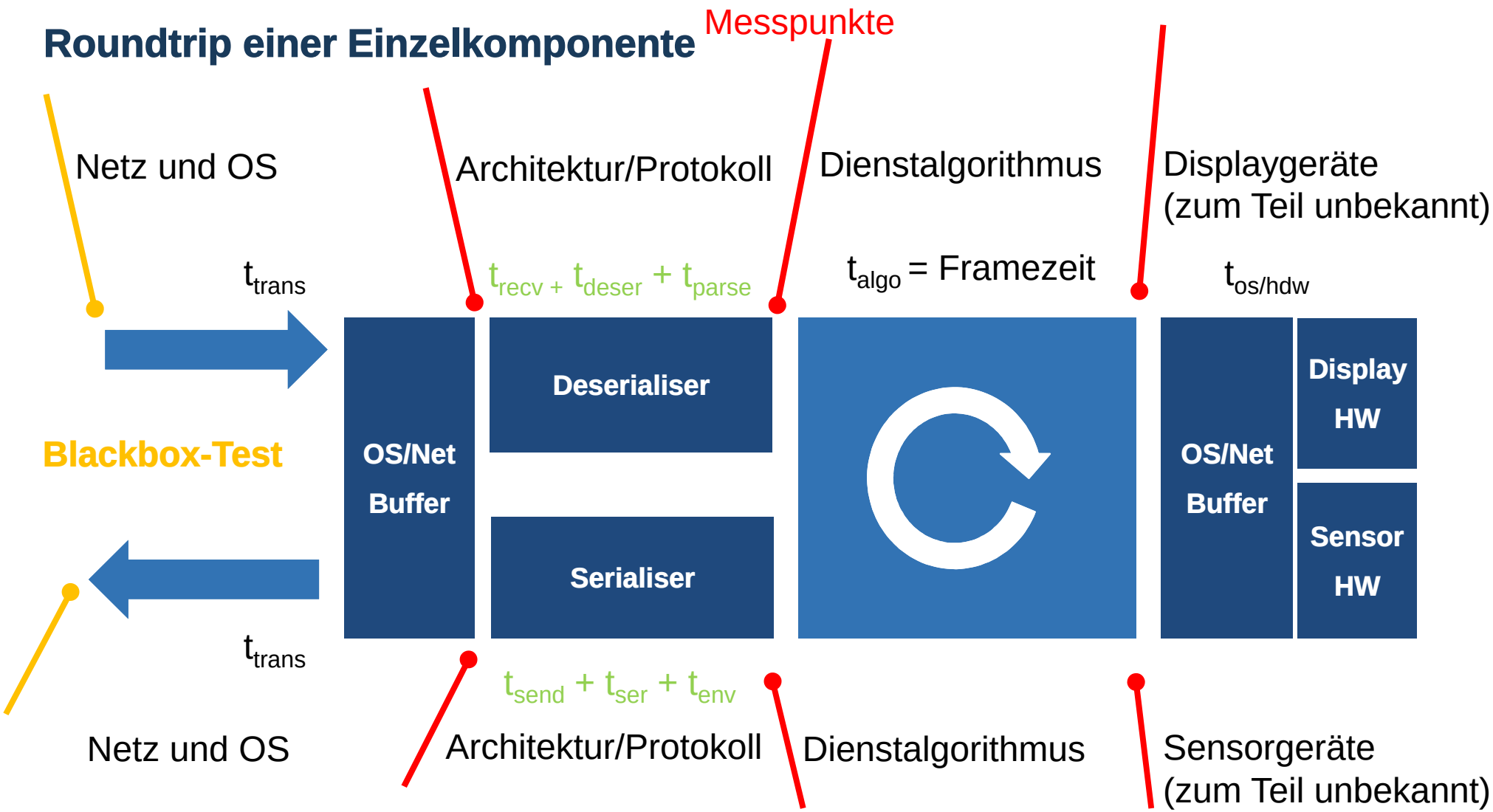
Architekturrelevante Zeiten

$t_{OS/HW} + t_{algo} + t_{write} + t_{ser} + t_{env} + t_{trans} + t_{deenv} + t_{deser} + t_{parse} + t_{algo} + t_{OS/HW}$



OS/HW | Dienst A | Dienst B | OS/HW

Roundtrip einer Einzelkomponente



Benchmark-Umgebung

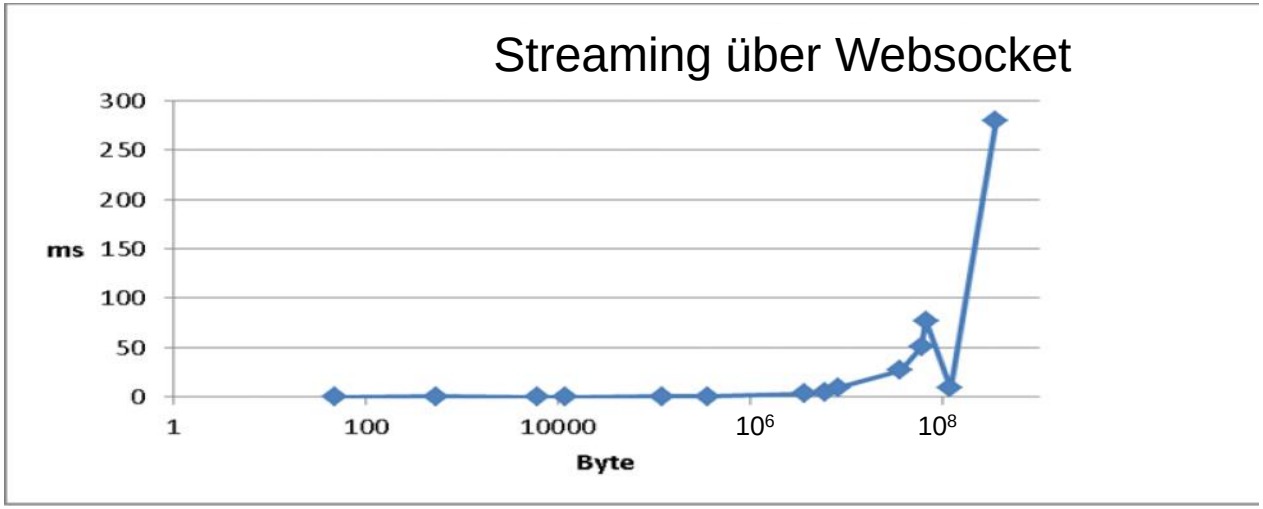
- Softwareumgebung
  - Basis Windows7 64Bit,
  - Microsoft Visual Studio 2013 C++, Compilerversion 18.00.31101 for x64
  - Arvida REST-SDK (C++)
    - Genutzte externe Komponenten: Boost, RAPTOR-RDF Framework, civetweb-HTTP-Server, curl-HTTP-Client
  - Minimale Dienste zum Test: Quell- und Senkendienst
- Hardware: Cluster aus 3 Knoten mit jeweils
  - 2 x Intel Xeon E5-2637 @ 3GHZ
  - Chipsatz C600/X79
  - 16 GB RAM, DDR3-1600
  - Graphik: 2 x Nvidia GTX 680
  - Netzwerkinterface: 1 x 10 Gbit Intel X540-T2 , 2x 1 Gbit Intel I350

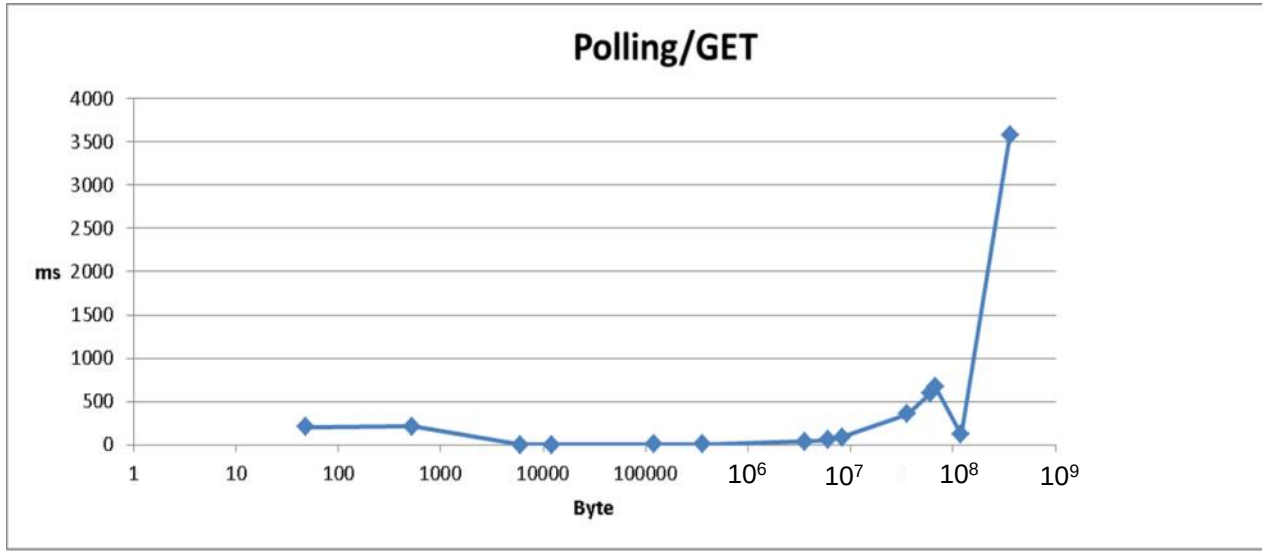
Use Cases (Testdatensätze)

- Trackerposen
  - **trck2**: Zwei Targets mit 6 Freiheitsgraden (6 DOF) = 2 x 6 x 4 Byte = 48 Byte*
  - **trckvh**: Ein Virtual Human 6DOF, 22 Targets = 22 x 6 x 4 Byte = 528 Byte
- 3D-Vektoren
  - **v3d500:** 500 3D-Vektoren, 500 x 3 x 4 Byte = 6.000 Byte
  - **v3d1000:** 1000 3D-Vektoren, 1.000 x 3 x 4 Byte = 12.000 Byte
  - **v3d10000:** 10000 3D-Vektoren, 10.000 x 3 x 4 Byte = 120.000 Byte
- 3D-Dreiecke
  - **tri10000:** 10000 3D-Dreiecke, 10.000 x 3 x 3 x 4 Byte = 360.000 Byte
  - **tri100000:** 100000 3D-Dreiecke, 10.000 x 3 x 3 x 4 Byte = 3.600.000 Byte
- Bilddaten
  - **hdimg:** HD-Kameradatenstrom, 1920x1080x32 Bit = 8.294.400 Byte
  - **4ktex:** Textur 4096x4096x32 Bit = 67.108.864 Byte

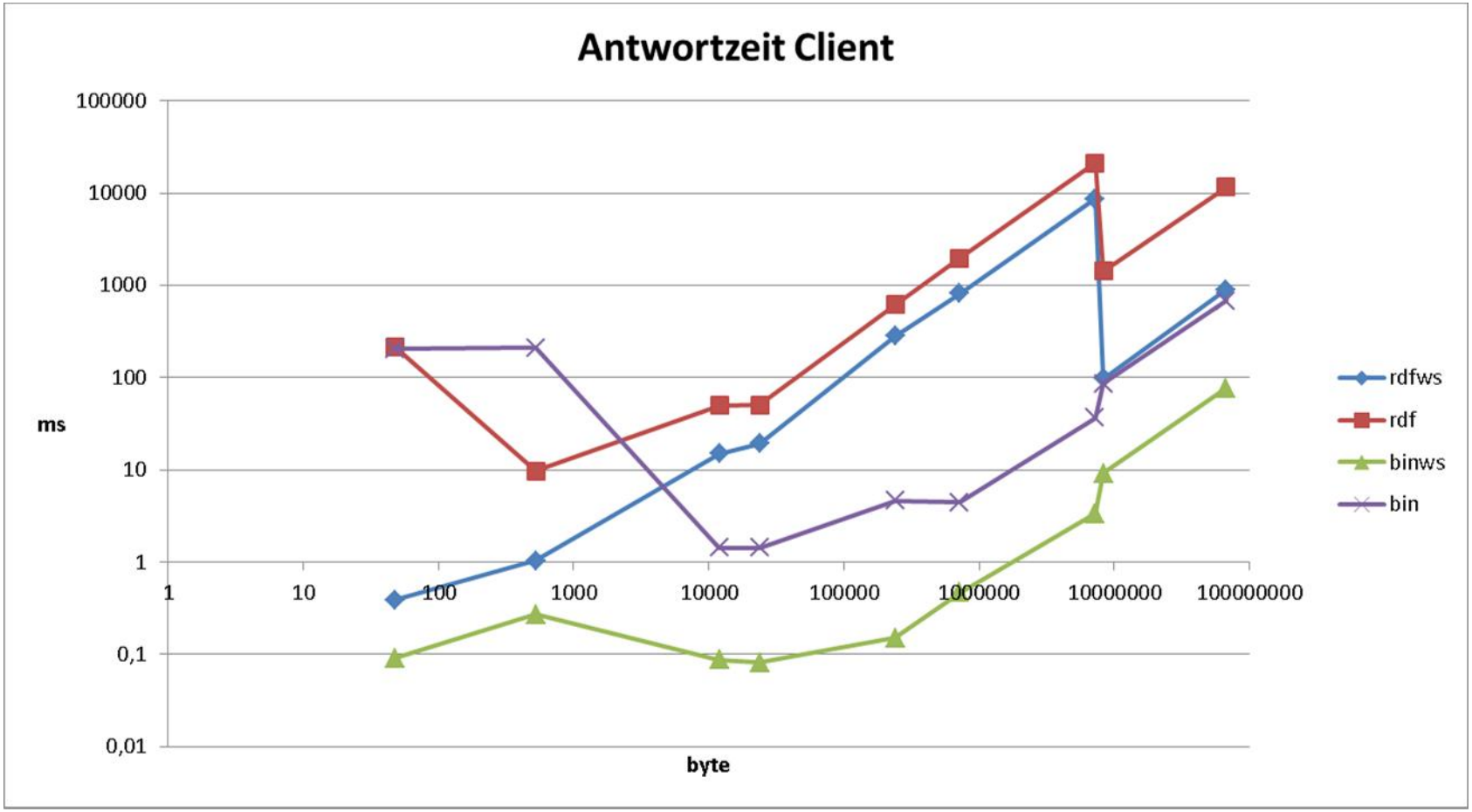
* Datenvolumen für single precision float

Ergebnisse: Vergleich Streaming mit Polling bei binärer Nutzlast

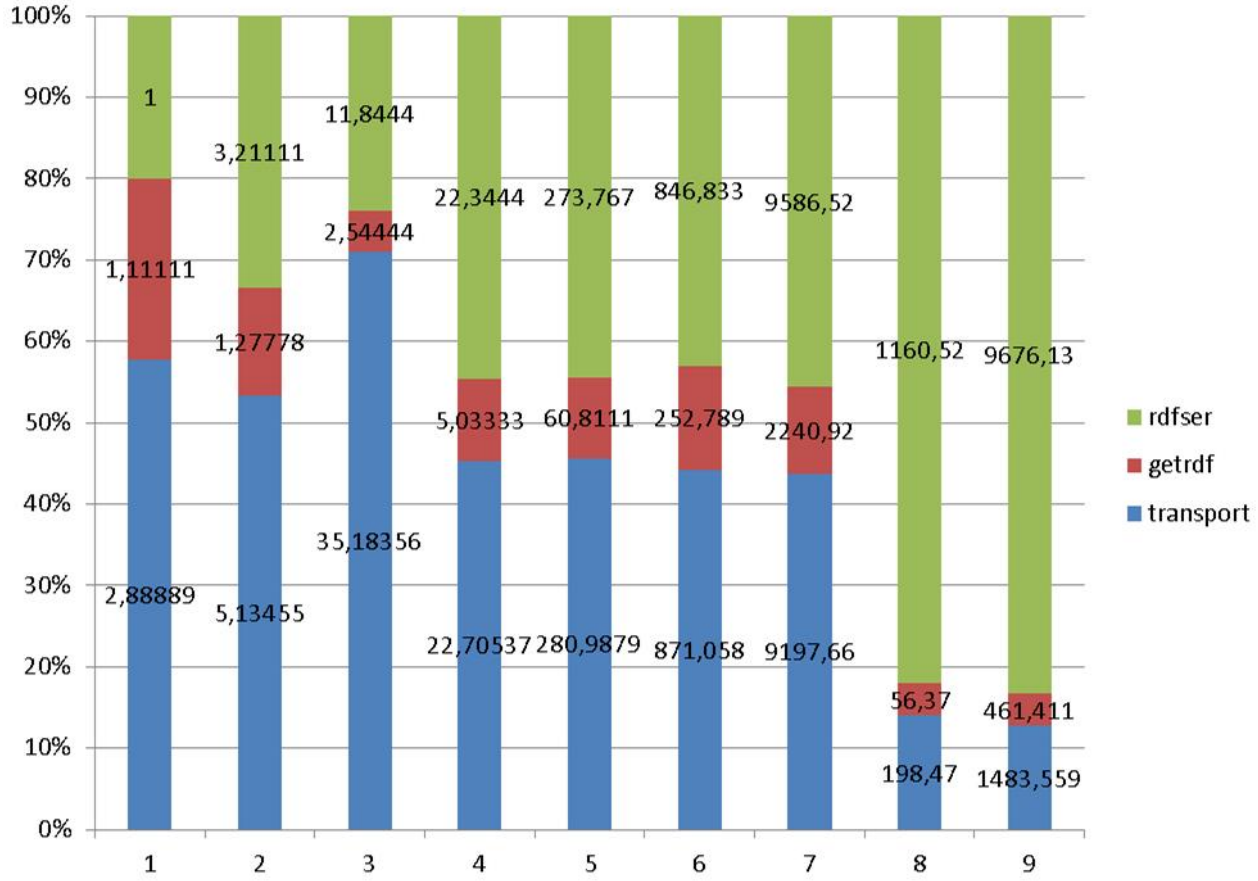




Ergebnisse: Vergleich binäre Nutzdaten mit RDF



Zeitbedarf der Einzelschritte (Client und Server, RDF plain)



trck2	48
trckvh	528
v3d500	12000
v3d1000	24000
v3d10000	240000
tri10000	720000
tri100000	7200000
hdimg	8294400
4ktex	67108864

(single precision)

- Flaschenhals RDF-Serialisierung
- Raptor2-Bibliothek vermutlich suboptimal

Zusammenfassung und Ausblick

- Evaluation ist wichtiger Teil des agilen Entwicklungsprozesses
- Messungen insbesondere geeignet zur Evaluation eingesetzter Protokolle und Komponenten sowie deren Implementierung
- Aufnahme der Zeitmessungen in Vokabulare als Komponente der Reflexion

Ausblick

- Nutzung von SPARQL (Graph-basierte Abfragesprache für RDF) zur automatischen Validierung und Optimierung
- HTTP/2 ist derzeit bester Kandidat für unterliegenden Technologiestack

Danke für Ihre Aufmerksamkeit!