VSS in-vehicle Performance

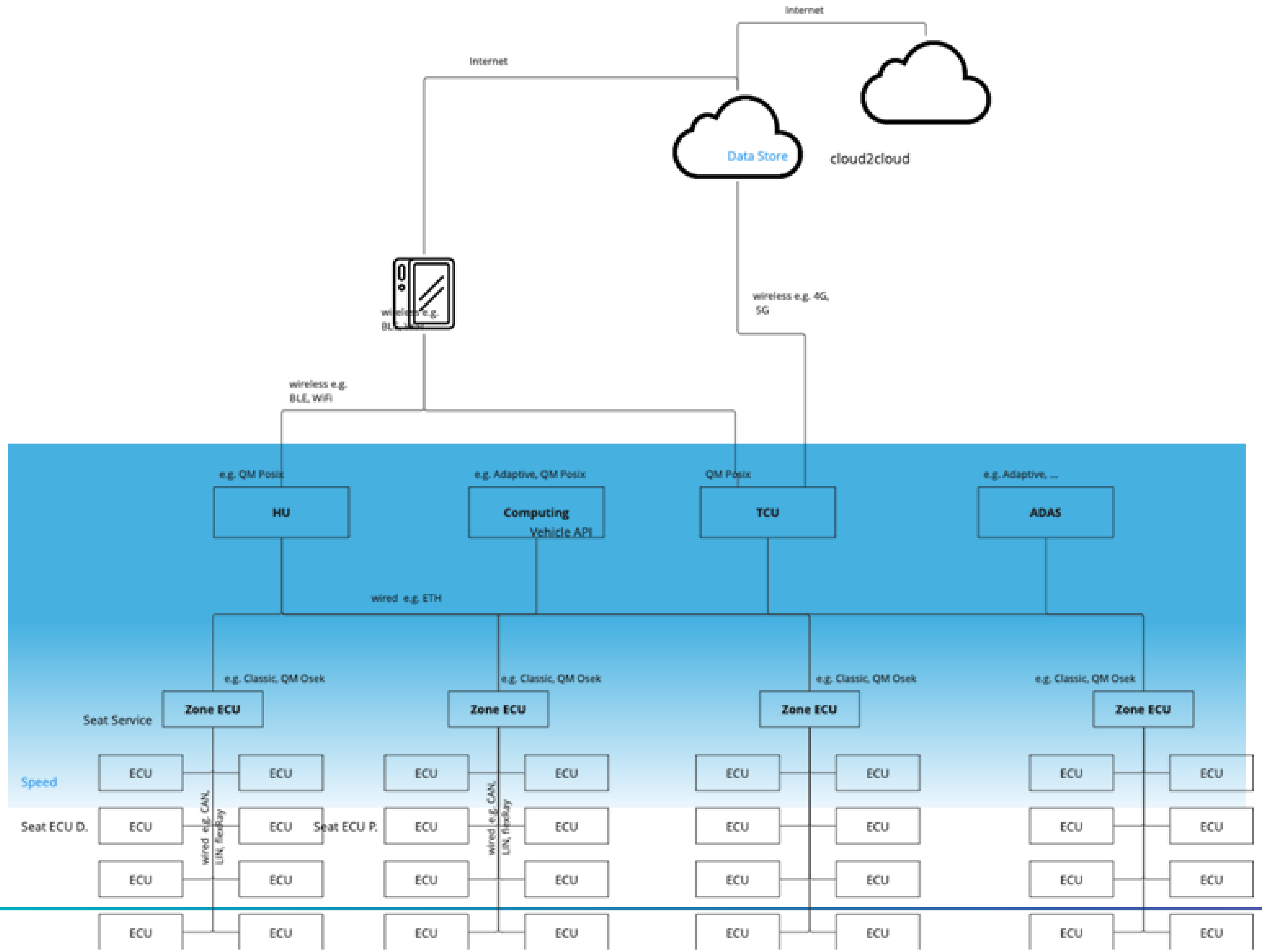
Considerations and impact on architecture

Sebastian Schildt, ETAS GmbH,
COVESA AMM, September 25th 2024

COVESA
Accelerating the future of connected vehicles

Copyright © 2024 COVESA

What we mean with “in-vehicle”



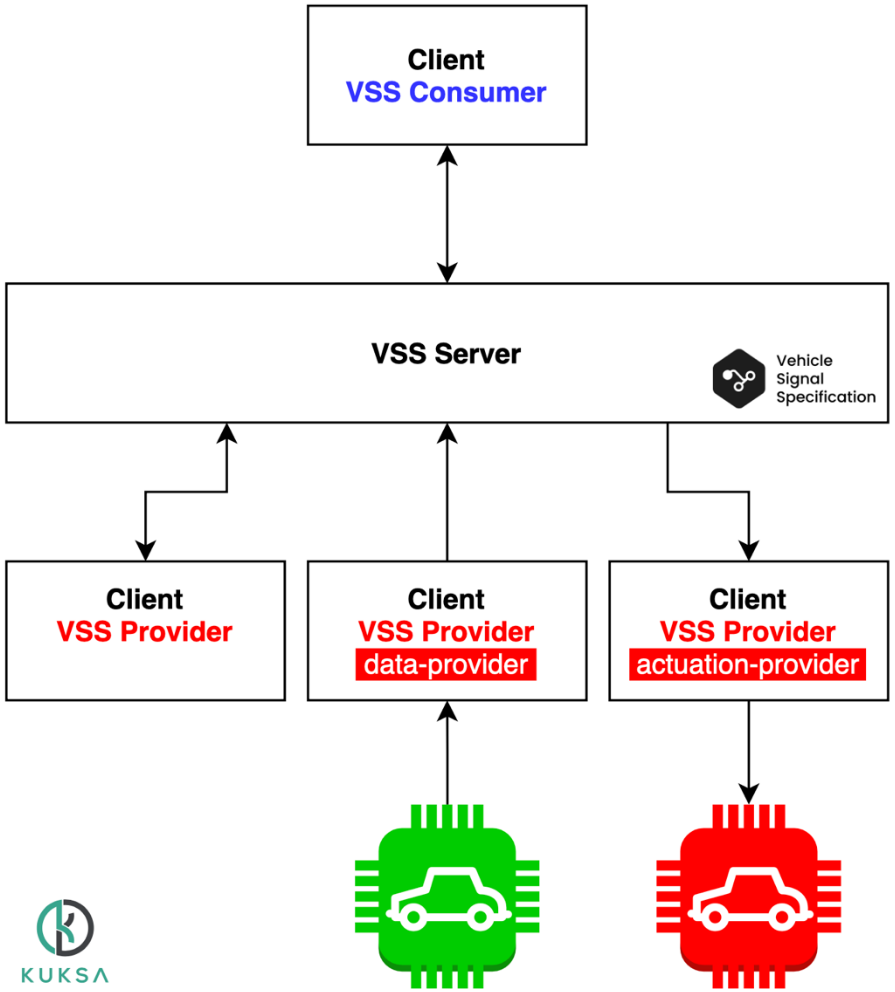
Compute units inside a car

- generate,
- consume and
- process

VSS data

COVESA

Taxonomy of in-vehicle VSS components



- Interacts with Vehicle represented by the VSS model
  - Vehicle Computer function
  - IVI App
  - External consumer device

- Holds current vehicle state in VSS format
- Provides an API to interact with VSS signals

- VSS provider syncs of the vehicle with VSS model of the server
  - **data-provider** makes sure that the actual state of a vehicle is represented in VSS (historically known as “feeder”)
  - **actuation-provider** makes ensure that the target value of a VSS actuator is reflected by the actual state of a vehicle

https://github.com/eclipse/kuksa.val/blob/master/doc/terminology.md

What happens in the stack

KUKSA

Application

Databroker
- Northbound API
- VSS Server
- Southbound API

API Design Space
Communication paradigm?
Target platform/languages?
Interaction pattern?

Providers
- Map/Convert to VSS
- Deserialize/Unpack
- Receive

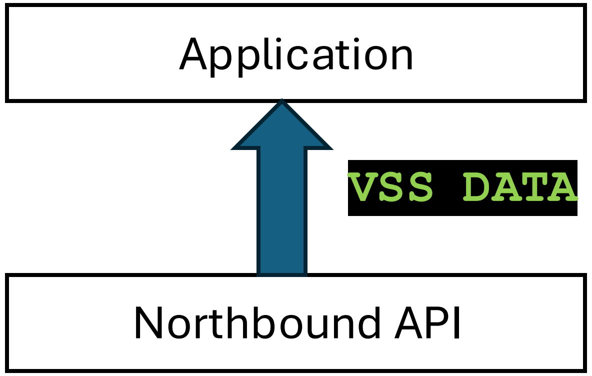
Dynamic/Static Code-generation/runtime

"Weather"

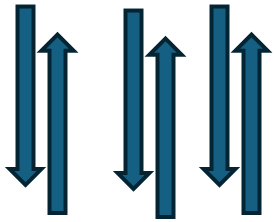
Not much to "do"/change here

Automotive busses

Communication paradigm



{ REST }

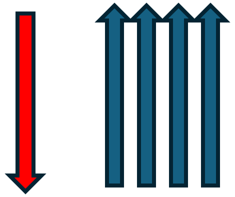


Query Response

- "Polling" for high frequency data not optimal
- Overhead as "state", e.g. security needs to reestablished #
- "Most" data in other vehicle systems is not using this

VISS

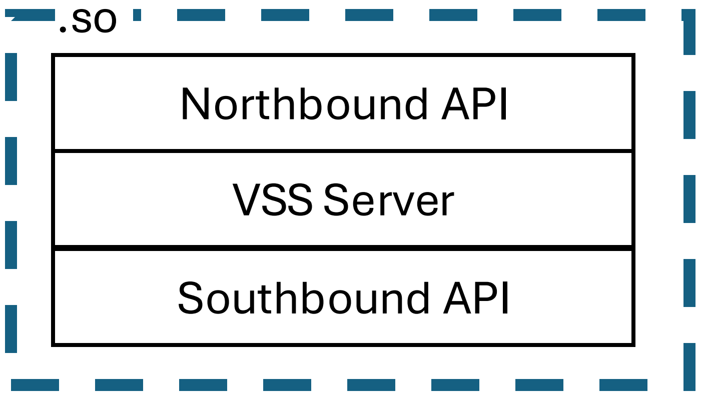
KUKSA



Publish Subscribe

- More efficient (less messages, state established once
- Fits patterns in embedded (e.g. CAN)
- Asynchronous nature can lead to challenges handling errors

To Link or Not to Link?

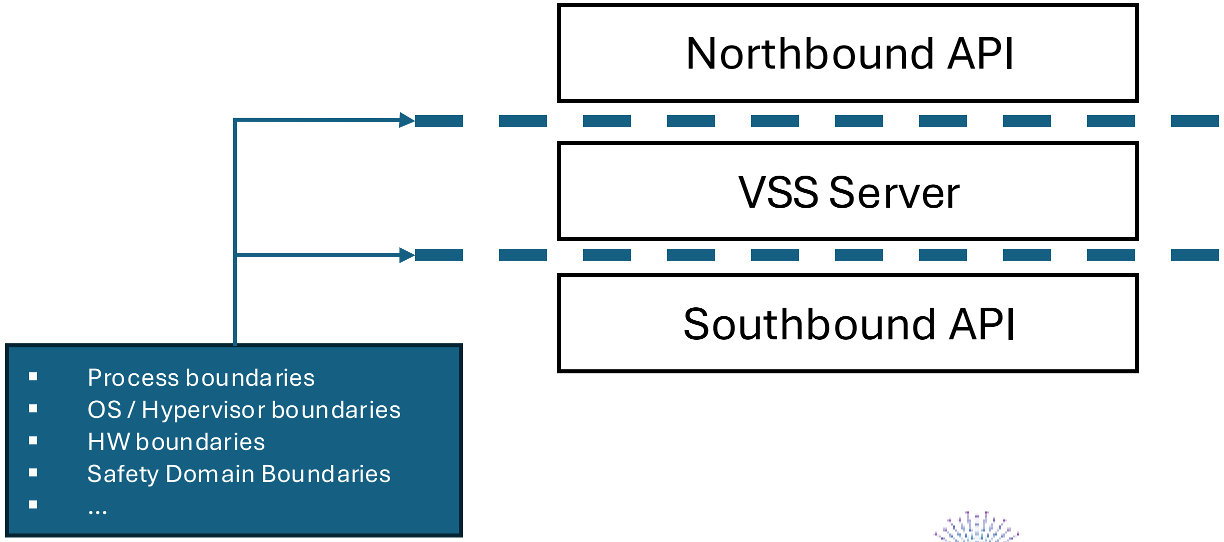


An “API” may just be a programming API that can be programmed against linked to.

Then the “serialisation”/data exchange becomes just a matter of the ABI of the platform

However, in modern vehicle systems/SDV systems we prefer **loosely-coupled** systems, often distributed (e.g. current E/E architectures are very distributed)

→ Likely looking into network APIs to cross system boundaries



Be faster: Shared Memory / Zero Copy

Zero copy a must for
- High bandwidth ADAS data
- Large volume (here (multiple) memcpy really hurt)

However, **there is a price**
- Tightly coupled systems
- Not easy between containers, compromising isolation and security
- Not really possible in systems distributed across the network*

Whereas **VSS data is often used** in
- Loosely coupled, ”SDV” systems
- Not always in a single trusted domain
- Distributed

* In “datecenter IT” there is RDMA/RoCE etc, but this is not scaled to Autpmtive style platforms AND doesn’t really prevent copy if somebody really NEED all the data (e.g. videostrms

Relax: Need for Speed?

Showing tire pressure every 5 seconds – Why even bother?

	Single Core	Multi-Core
Laptop M3	3138	14128
iMX8 (Cortex A53)	188	557
Pi 4 (Cortex A72)	290	657

For applications: Yes

For VSS middleware: Used by ALL applications – has an impact

Do not overestimate speed of modern Vehicle Computers

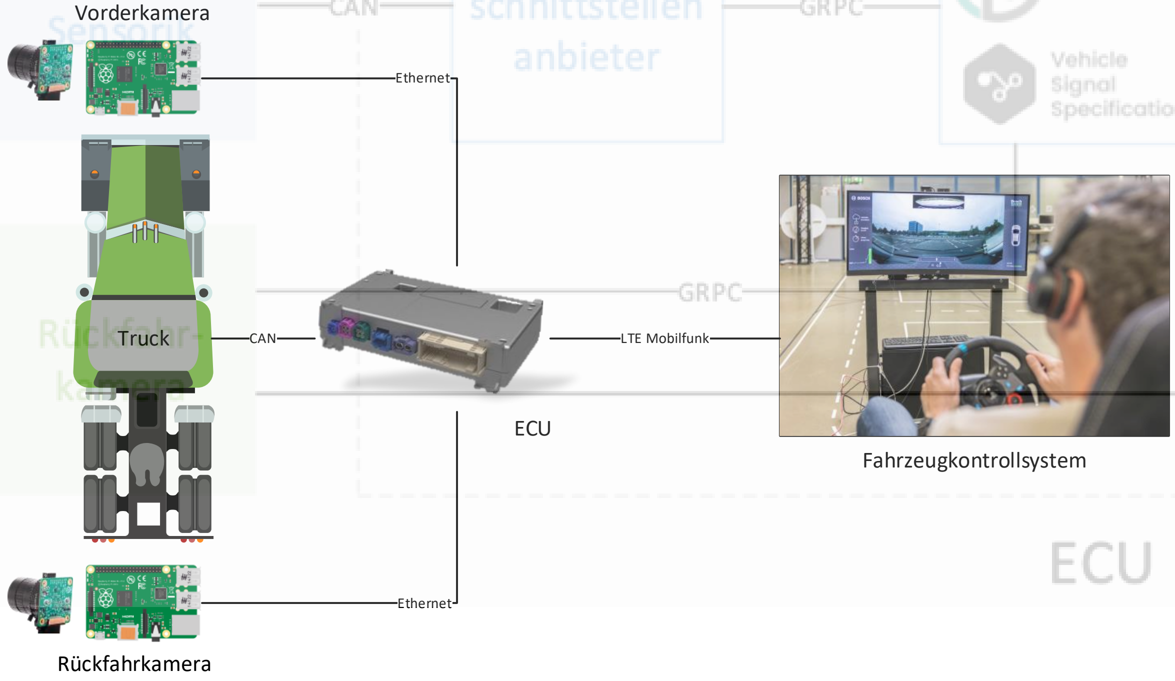
Want to serve not only high end, but also mid/low

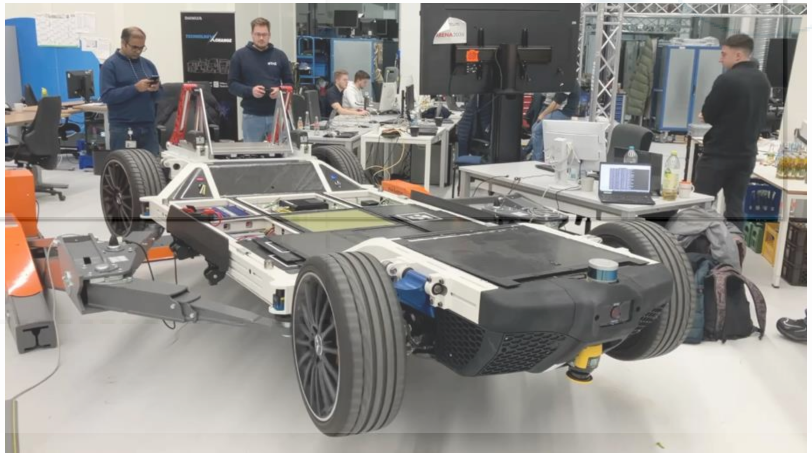
High Frequency for preprocessing (crash detection, driving scores, tire state prediction,...)

Example: High frequency actuation

Remote control from SofDCar Research project

- Motion control based on a vehicle-independent VSS model
- Running on a production telematic unit
- User experience and safety (regulations!) depend on low cycle times





Tech Choice GRPC

One good base technology for VSS data in a vehicle is GRPC

Efficient Serialization: gRPC uses Protocol Buffers (Protobuf) as the default serialization format, compact, **strongly typed** and fast (e.g. compared to JSON) and proven

HTTP2: gRPC uses HTTP/2 as the underlying transport protocol, allowing for multiplexing requests and responses over a single connection, reducing overhead and improving performance.

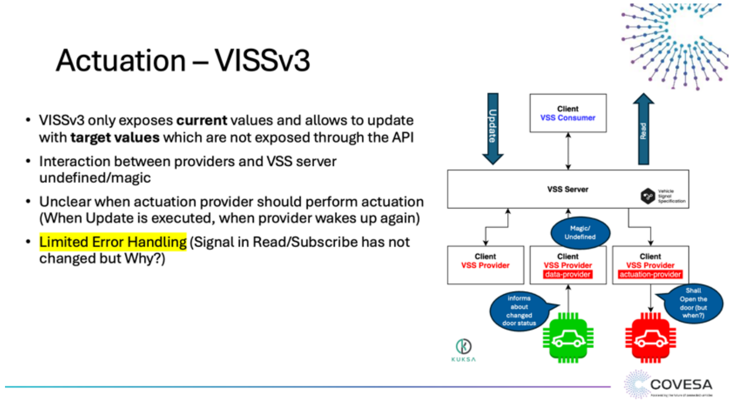
Language Agnostic gRPC supports multiple languages, in theory AND practice including but not limited to Rust, C++, Java, Python, Go, C#.

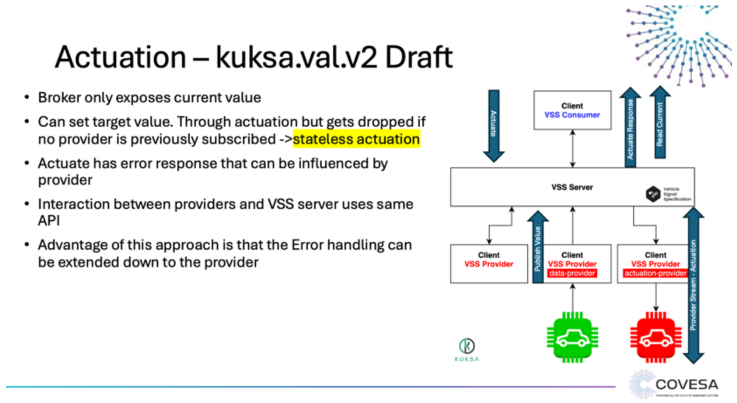
Bi-Directional Streaming gRPC supports four types of APIs: unary (single request-response), server streaming, client streaming, and bi-directional streaming. This allows for building efficient applications where both clients and servers can send a continuous flow of data.

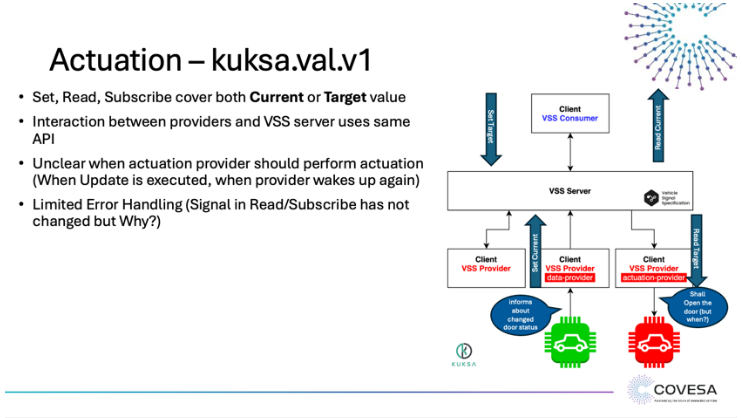
VISS-GRPC option

The details, the details!

"Let's use GRPC and PubSub" can still lead to different approaches







remotiveLabs

BlackBerry IVY

?

SONATUS

...

Benchmarking

Tech choices bring you in “order of magnitude” target range, the last 2x/3X difference is the result of “engineering”

- Benchmarking is hard - in an use case need to know the End-to-End performance.
- Having solid benchmarks of individual components is a good first step

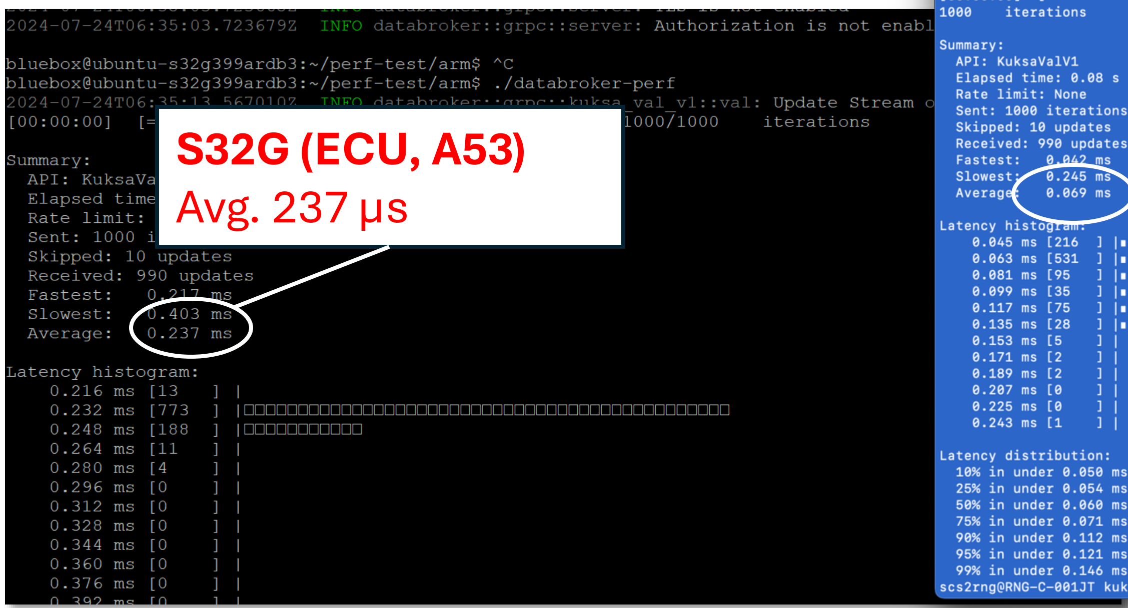
In any case tread carefully, and take any results here or elsewhere with a grain of salt. Or as a DE German engineer would put it:

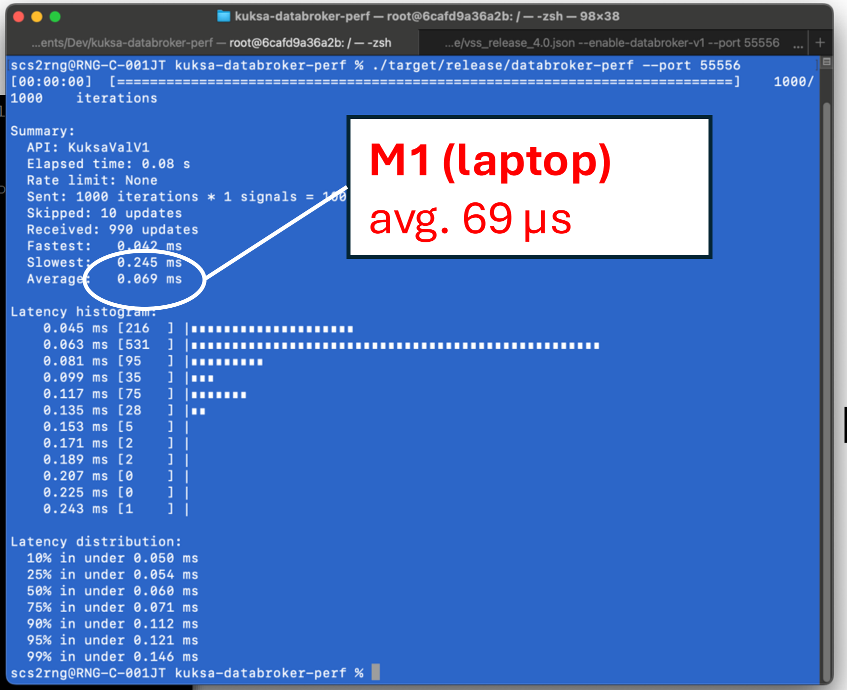
Wer misst, misst Mist!

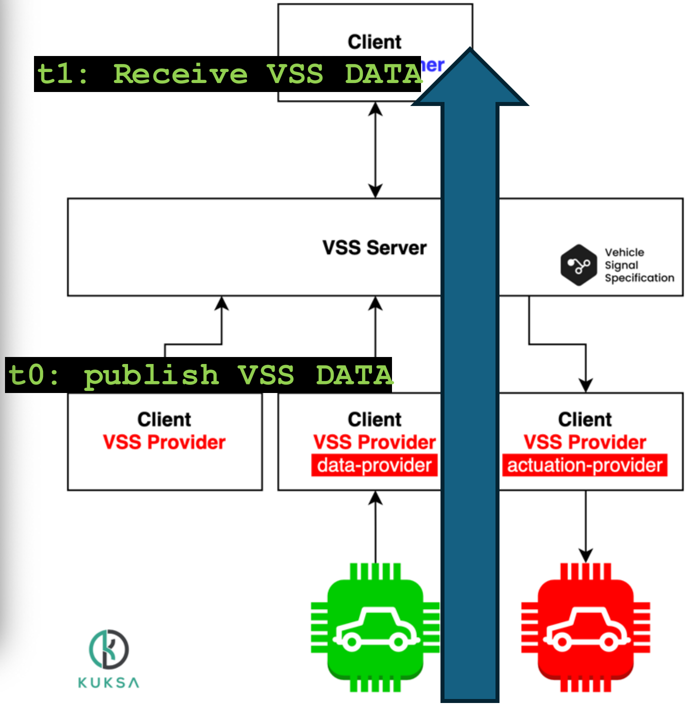
kuksa-perf

Made first ***synthetic*** benchmark of KUKSA API publicly available

While maybe not indicative of real-world performance it gives a repeatable base value: If this is below your app requirements, this is not the right software for you or you have chosen the wrong hardware





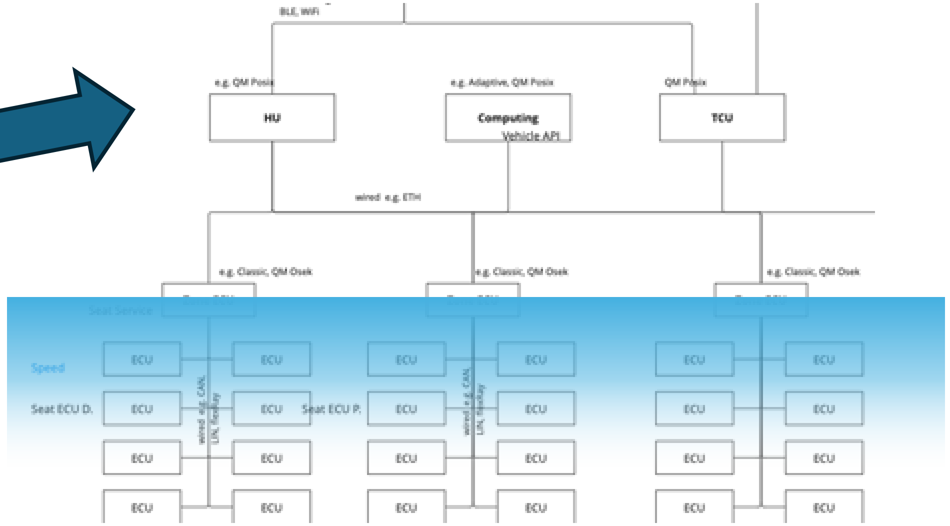


https://github.com/eclipse-kuksa/kuksa-perf

Go lower

KUKSA

- Run reasonable on anything with a processor and a POSIX OS
- Fast without sacrificing developer productivity



- A lot of ECU running in µC using small RTOS/AUTOSAR classic systems
- GRPC already considered “heavy”*
- Might not even have/want an IP stack

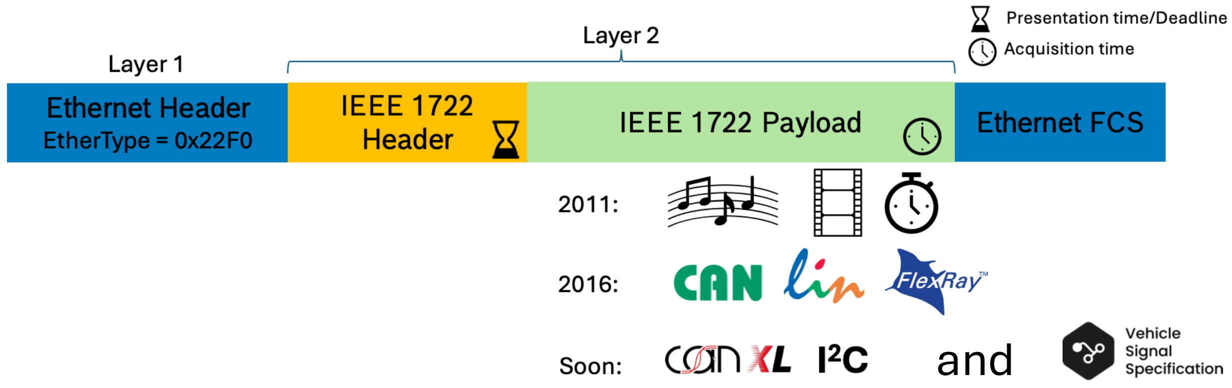
No VSS here

?

* Can be done though, see GRPC based Provider on Espressif ESP32: https://github.com/eclipse-kuksa/kuksa-incubation/tree/main/gRPC-on-ESP32

Enter IEEE 1722 & Open1722

An efficient Ethernet L2 (UDP optional) transport protocol



Join Open1722 talk tomorrow 2pm for details!

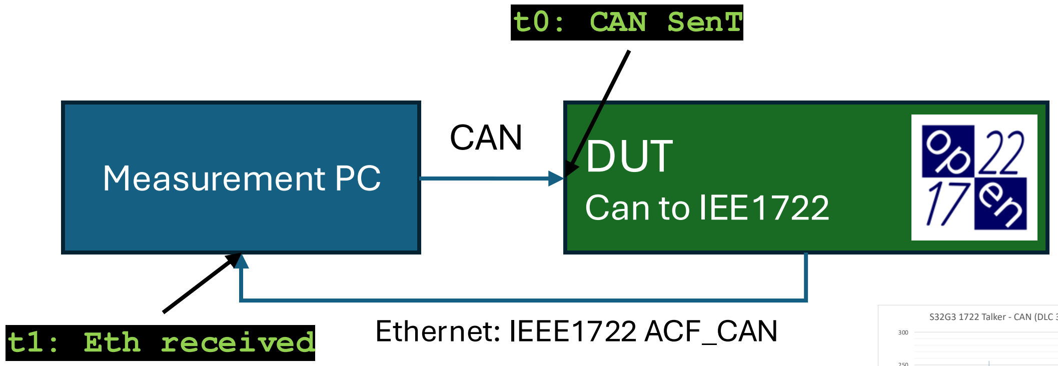
<u>Executive Summary</u>

This runs easily on a µC

Does not need Linux/POSIX

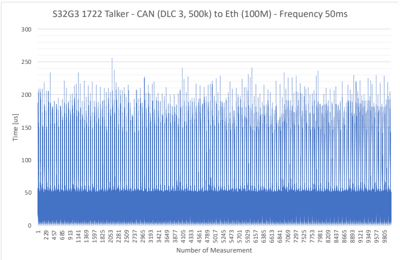
Turns out can be easily adapted to support VSS natively

Open1722 performance



S32G
(CortexA53)
+ Ubuntu

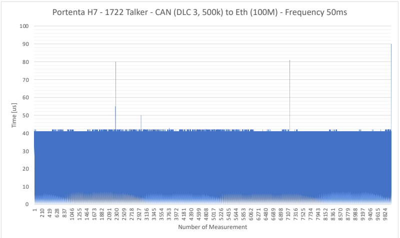
Avg: 63.6 uS



Zephyr

Portenta H7
(CortexM7)
+ Zephyr

Avg: 41 uS




```
    Padding: 000000
ACF Message: Reserved (0x42)
  ACF Header: Reserved (0x42), 28 bytes with header
  ACF VSS Header
    01.. .... = VSS Padding: 0x1
    ..0. .... = VSS Timestamp Valid: 0x0
    ...0 0... = VSS Addressing Mode: INTEROP_MODE (0x0)
    .... .000 = VSS Opcode: TARGET VALUE (0x0)
    VSS Datatype: UTF8-STRING (0x0b)
    VSS Timestamp: Jan  1, 1970 00:00:00.000000000 UTC
  VSS Path
  VSS Data
    VSS Data: World!
    Padding: 00
ACF Message: Reserved (0x42)
  ACF Header: Reserved (0x42), 28 bytes with header
  ACF VSS Header
```

Open1722 can also send VSS data

This is _just_ transport. No broker/server no access control, no “returns & error”

Same performance possible using ACF_VSS,
This is probably the best you can get performance –wise transmitting VSS data in a vehicle

COVESA
Accelerating the future of connected vehicles

Summary & Final thoughts (1/2)

VSS in vehicle is cool, but need to ”aim carefully” with tech stack

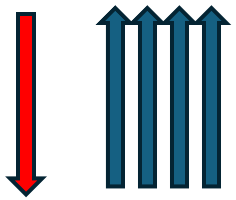
- What use cases to support?
- How “deep” in the E/E architecture you want to use it?
- How “wide” you want to serve the market?

  Mid/entry level architectures WILL have processors,

  just not the 32core Qualcomm + 64GiB of RAM….

Summary & Final thoughts (2/2)

- Some things are “aligned” / evolved in parallel
    - No question that PubSub is the way to go
    - GRPC seeing general adoption in Automotive
- Open1722+VSS is a cool convergence technology to do for ”VSS” what CAN did for, well bits & bytes
- In terms of efficient VSS in-vehicle APIs ther might be room for a (COVESA) standard



COVESA

Accelerating the future of connected vehicles

COVESA VSS	Vehicle Signal Specification	https://covesa.github.io/vehicle_signal_specification/
/me		http://sdv.expert
KUKSA	KUKSA	https://eclipse.github.io/kuksa.website/
Examples		https://wiki.covesa.global/
ETAS OSS	ETAS	https://www.etas.com/en/open-source-software.php

Copyright ©2024 COVESA