# A QoS Aware Approach to Service-Oriented Communication in Future Automotive Networks

Mehmet Çakır, Timo Häckel, Sandra Reider, Philipp Meyer, Franz Korf and Thomas C. Schmidt
*Dept. Computer Science*, *Hamburg University of Applied Sciences*, Germany
{mehmet.cakir, timo.haeckel, sandra.reider, philipp.meyer, franz.korf, t.schmidt}@haw-hamburg.de

***Abstract*—Service-Oriented Architecture (SOA) is about to enter automotive networks based on the SOME/IP middleware and an Ethernet high-bandwidth communication layer. It promises to meet the growing demands on connectivity and flexibility for software components in modern cars. Largely heterogeneous service requirements and time-sensitive network functions make Quality-of-Service (QoS) agreements a vital building block within future automobiles. Existing middleware solutions, however, do not allow for a dynamic selection of QoS.**

**This paper presents a service-oriented middleware for QoS aware communication in future cars. We contribute a protocol for dynamic QoS negotiation along with a multi-protocol stack, which supports the different communication classes as derived from a thorough requirements analysis. We validate the feasibility of our approach in a case study and evaluate its performance in a simulation model of a realistic in-car network. Our findings indicate that QoS aware communication can indeed meet the requirements, while the impact of the service negotiations and setup times of the network remain acceptable provided the cross-traffic during negotiations stays below 70% of the available bandwidth.**

***Index Terms*—In-vehicle communications, Quality-of-Service, Service-Oriented Architecture, Middleware, Automotive Ethernet**

## I. Introduction

Modern cars are evolving toward software-driven cyber-physical systems with ever-growing communication demands. The subsystems of emerging vehicular architectures are increasingly interconnected to create advanced, often complex functions. These cross-domain functions undermine previous hard boundaries between the different automotive software domains. The integration of new components thereby becomes more difficult as dependencies and interactions are richer and harder to predict [1]. Besides, opening up the car network to the environment (Car-to-X) requires new communication technologies such as secure and dynamic restful services.

The communication architecture of modern vehicles has become so complicated that it tends to stop innovation rather than promote it [2]. According to a broad study within the automotive industry conducted by fortiss [2], these problems can only be solved by redesigning the automotive communication architecture. A high bandwidth communication backbone is proposed, in which software components communicate in a service-oriented manner [1].

Automotive Ethernet has emerged as the next high-bandwidth communication technology for in-car backbones [3]. Complementary protocols such as IEEE 802.1Q Time-Sensitive Networking (TSN) [4] provide QoS-guarantees and have proven to meet the real-time and robustness requirements of the automotive environment [3]. With the inclusion of Scalable service-Oriented MiddlewarE over IP (SOME/IP) [5], AUTOSAR paves the way for service-oriented, IP-based communication in automotive systems. SOME/IP focusses on implementing SOA while still supporting small Electronic Control Units (ECUs). Currently, a mechanism is missing that merges the concepts of SOME/IP and QoS-enhanced communication for dynamically changing communication relations.

In this work, we contribute a QoS Negotiation Protocol, which allows services to dynamically achieve QoS agreements. Based on the QoS requirements of automotive services, a classification into four QoS classes is derived. A multi-protocol stack is introduced that satisfies the heterogeneous communication demands of automotive applications. We propose a framework for a QoS aware service-oriented middleware that is tailored to the requirements of a car. Thereby we focus on deadlines and latency as the essential QoS measures of real-time systems. In a subsequent case study, we analyze whether the automotive requirements can be met. Evaluations are performed with the help of an automotive simulation environment in OMNeT++ [6]. Communication efforts caused by the middleware are quantified using realistic vehicle communication.

The remainder of this paper is structured as follows. Section II revisits background technologies and related work. Section III analyses requirements to derive a classification of automotive services into four categories. The design of our middleware and the QoS negotiation are presented in Section IV. Section V discusses the case study, in which the performance of the middleware is evaluated by simulations. Finally, Section VI concludes with an outlook on future work.

## II. Background and Related Work

Ethernet has been recently extended in IEEE 802.1Q-2018 [4] by Time-Sensitive Networking (TSN), which defines a set of primitives to gain the ability of forwarding real-time- and cross-traffic concurrently. To support a variety of QoS requirements, TSN provides several real-time traffic classes. These can be synchronous (Time Division Multiple Access (TDMA)) or asynchronous such as TSNs predecessor Audio Video Bridging (AVB), which we analyzed in previous work [7].

This work is funded by the Federal Ministry of Education and Research of Germany (BMBF) within the SecVI project.

AUTOSAR[1] is one of the key players in the standardization of automotive communication solutions as most ECUs are based on this platform. Gopu et al. [8] emphasize that AUTOSAR takes the challenge of finding solutions between legacy support and innovation. With the introduction of Scalable service-Oriented MiddlewarE over IP (SOME/IP) [5] AUTOSAR was augmented by service-oriented, IP-based communication in automotive systems. The SOME/IP protocol consists of several components that solve problems such as data transportation, serialization, and service discovery. SOME/IP focusses on implementing a SOA while still supporting small ECUs. SOME/IP already supports static QoS with IEEE 802.1Q priorities, however, dynamic QoS negotiations to establish real-time communication are not supported. Even though we do not explicitly confine our approach to the SOME/IP protocol, the developed QoS negotiation process can be applied to the SOME/IP middleware layer service connection process. Whenever a new client requests information at a server, the negotiation procedure presented in this work could be applied for ongoing connections.

Besides SOME/IP, there are many other experimental middleware solutions in the automotive domain and for cyber-physical systems [9]–[14], which mainly focus on implementing IP-based communications. In contrast, our work addresses a middleware approach that supports heterogeneous QoS requirements, which is indispensable for future backbone and zonal architectures in automotive networks.

Due to the substantial similarity of the sensor actuator systems of a vehicle with those of industrial plants, concepts from this field of research can also be transferred to cars. Cucinotta et al. [15] as well as Jammes et al. [16] introduce service orientation to automation technology. They present similar requirements as those of in-car communication architectures. Cucinotta et al. developed the software platform *RI-MACS* with a split stack of communication protocols for different classes of services, which are provided and controlled by a QoS-based middleware. We follow a similar approach concerning the negotiation and the automotive-specific implementation.

Menascé et al. [17] emphasize that QoS-aware middlewares are best to satisfy as many consumers as possible in networks of components with differentiated requirements, specifically in highly distributed service-oriented systems. In their work, they present a protocol for negotiating performance characteristics. Negotiations are accomplished via a central QoS-Broker, which serves as an intermediary between the services. A client negotiates the performance characteristics for a flow with the responsible broker of the service provider. In contrast to this centralized approach, we distribute the broker function between the provider node and the client-side.

Abdelzaher et al. [18] apply QoS-based service-oriented communication in the real-time system of airplanes and focus on ensuring functionality and real-time for the key components. In their architecture, services communicate their requirements and are then assigned shares of a statically defined resource pool. Whenever the requested resources cannot be reserved, a process of *graceful degradation* begins. Rather than denying communication to services, some services are selected according to priorities and downgraded in QoS. Hence, the communication for the high-priority components can be guaranteed even in overloaded situations. The QoS negotiation we present could be augmented by such degradation procedure, which might be a suitable solution for future autonomous driving scenarios.

[1]AUTomotive Open System ARchitecture: https://www.autosar.org/

Becker et al. [19] describe a runtime monitoring framework based on the SOME/IP middleware layer. They focus on admission control, monitoring, and adaption to enforce the prescribed behavior of software components to minimize the impact of misbehavior on other flows. This is only possible for well known components with well defined behavior and will not be enough in dynamic scenarios. To guarantee QoS aware communication for real-time applications this behavior must be enforced on the network level. By introducing QoS agreements based on client requirements, it is possible to provide such guarantees even for dynamically changing communication relations.

Another approach to QoS based middleware solutions is Data Distribution Service (DDS) from the OMG [20]. Again, QoS parameters are used to control the communication properties of endpoints. In the network, service providers are represented by a publisher and consumers by a subscriber. The data is passed on to the publisher via a writer and received by the subscriber via a reader. Disconnecting the publisher from the service itself allows a writer to serve multiple publishers. Although the publish/subscribe pattern is very suitable for communication in the car, DDS has a significant overhead and is not compatible with existing automotive communications. This work combines the different approaches including the QoS based publish/subscribe approach of DDS and makes it compatible with low-level communication such as SOME/IP to create a service-oriented middleware solution for heterogeneous applications in the automotive domain.

## III. Requirements Analysis for Automotive Service Classes

Automotive services are heterogeneous with differentiated QoS requirements for communication— all of which must be supported by a future automotive middleware solution [1]. Guided by these QoS requirements, we divide the services into classes, which then define a specific treatment by the middleware. An overview of the different criteria for service classification is given in Table I. This table shall help in identifying the correct QoS class for a specific software component.

### A. Criteria

The *Automotive Software Domain* defines one category to classify groups of similar automotive software components that is most commonly used in the literature [21]–[23]. According to Pretschner et al. [23], software of the domains

TABLE I: Requirement driven classification of automotive software associated with our four QoS classes: Static Real-Time Services (SRTS), Real-Time Services (RTS), IP-based Services (IPS), Web Services (WS).

| **Criterion** | **Source** | **SRTS** | **RTS** | **IPS** | **WS** |
|---|---|---|---|---|---|
| **Software Domain** | [21] [22] [23] | Safety Electronics, Engine/Powertrain | Safety Electronics, Engine/Powertrain, *(Multimedia/HMI)* | all domains | Multimedia/HMI, Passenger/Comfort, Diagnostics/Infrastructure |
| **Level of Abstraction** | [24] | Signal/Physical | Data, Signal/Physical | all levels of abstraction | Behavior, Knowledge, Information |
| **Realtime Constraints** | [24] | Simple Control Loops | Vehicle Dynamics Control, Simple Control Loops | Mission Control, Tactical Control | Cooperative Control, Mission Control |
| **Locality Constraints** | [24] | Aggregates, Sensors and Actuators | Vehicle, Aggregates, Sensors and Actuators | Vehicle, Aggregates, Sensors and Actuators | Global Environment, Local Environment, Vehicle |
| **Performance of the Environment** | [22] | Micro-Devices | PCs, Micro-Devices | PCs, Micro-Devices | Cloud-Infrastructure, PCs |

differs in its requirements for communication deadlines, data complexity, and communication patterns. The most commonly described domains and requirements are:

- *Multimedia/HMI* – Deadline: Soft, range of $100-250\,ms$; Data Complexity: Complex with large data structures; Communication Pattern: Discrete events and streams, off-board interface.
- *Passenger/Comfort* – Deadline: Soft, range of $100-250\,ms$; Data Complexity: Mixed; Communication Pattern: Discrete event processing dominates over control programs.
- *Safety Electronics* – Deadline: Hard, range of $1-10\,ms$; Data Complexity: Low, single values; Communication Pattern: Discrete event-based, strict safety requirements.
- *Engine/Powertrain* – Deadline: Hard, range of micro to milliseconds; Data Complexity: Low, single values; Communication Pattern: Control algorithms dominate over discrete event processing, strict availability requirements.
- *Diagnostics/Infrastructure* – Deadline: Soft and hard; Data Complexity: Mixed; Communication Pattern: Event-based software for management of the IT systems in the vehicle.

Jobst et al. [24] name three different approaches to layering automotive services based on different criteria: *Level of Abstraction*, *Temporal Layers*, *Spatial Layers*.
The *Level of Abstraction* describes the abstraction and responsibility associated with a service. It focuses on the functionality and information provided to other layers and gives insight into the nature of the data to be transferred:

- *Behavior* – e.g., traffic control, swarm coordination
- *Knowledge* – e.g., “car $25\,\mathrm{m}$ ahead”
- *Information* – e.g., “object $25\,\mathrm{m}$ ahead” and “car ahead”
- *Data* – e.g processing of individual values
- *Signal/Physical* – e.g., raw measurements, actuator control

The *Temporal Layer* describes the timing requirements according to the cycle time that passes until the next execution. The cycle time limits the maximum latency of messages as otherwise, the information may be outdated. The maximum latency allowed for messages in the automotive domain is approximately 10% of their cycle time. Low cycle times of only a few milliseconds are especially susceptible to jitter and latency [25]. The following describes the different temporal layers:

- *Cooperative Control* – e.g., coordinating swarm behavior, cycle time: unlimited
- *Mission Control* – e.g., computing way points, cycle time: $50\,\mathrm{ms}$
- *Tactical Control* – e.g., computing trajectories, cycle time: $10\,\mathrm{ms}$
- *Vehicle Dynamics Control* – e.g. pursuing trajectories, cycle time: $5\,\mathrm{ms}$
- *Simple Control Loops* – e.g., motor and actuator control, cycle time: $1\,\mathrm{ms}$

The *Spatial Layer* (see [24]) describes where (geographically) within the communication network a service is located, which provides information about the appropriate communication protocols. By opening the vehicular network to the Internet and introducing cloud computing approaches into the car, the location of a service becomes an important criterion:

- *Global Environment* – e.g., city-wide traffic
- *Local Environment* – e.g., a swarm of five cars
- *Vehicle* – e.g., ECUs in specific spatial zones
- *Aggregates* – e.g., motor control
- *Sensors and Actuators* – e.g., temperature sensor

Schäuffele et al. [22] identify the *Performance of the Environment* as another key criteria. This describes capabilities of the hardware components that run the service. The performance is divided into the following groups:

- *Cloud-Infrastructure:* Scaling environments of computers with virtually unlimited performance, supporting Internet-based protocols and providing sufficient processing power for complex operations.
- *PCs:* Computers that can host a full Linux with potent hardware. They support Internet-based and local protocols and provide sufficient processing power for complex operations.
- *Micro-Devices:* Microcomputers with minimal resources. They support local communication interfaces but are (without hardware modules) unable to implement complex operations, such as encryption or complex serialization with reasonable effort.

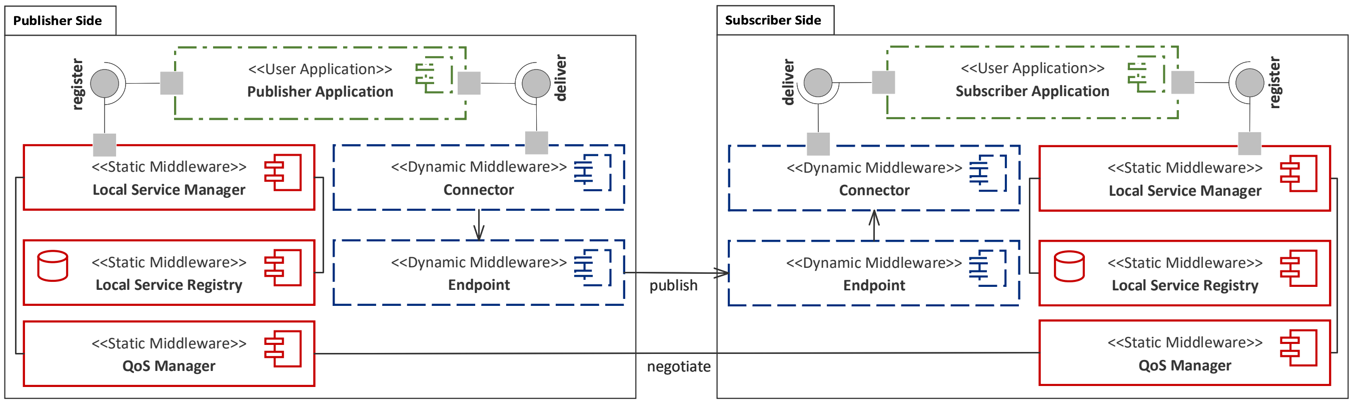

Fig. 1: Middleware architecture overview containing middleware components and the communication path.

### B. *Classification*

We now derive different service classes from the criteria described above, each of which represents a service requester group to the communication infrastructure. The characteristics and criteria may overlap in some areas, but each class focuses on unique criteria. In this work, we have chosen four service classes Static Real-Time Services, Real-Time Services, IP-based Services, and Web Services, which are depicted in Table I.

The *Static Real-Time Services (SRTS)* class groups various signal-based control loops that are safety-critical and highly prone to jitter and latency. Messages from these services need to be transmitted in a scheduled manner. For this reason, SRTS services are not dynamically usable. Still, they can be represented as a dynamic service to the network via a gateway.

The *Real-Time Services (RTS)* class focuses on time-critical, internal communication with hard deadlines. RTS are executed on PCs and Micro-Devices. The core domains in this class are Safety Electronics and Engine/Powertrain. Multimedia streams have real-time requirements as well, but they do not have the same time scale and can be softened by mechanisms such as buffering, which is impracticable for sensor data. For implementation in the middleware, Ethernet-based real-time transmission protocols must be used, which allow resource allocation during runtime.

The *Web Services (WS)* class groups all services that focus on a high level of abstraction and global offer with no or only soft deadlines. Due to the global offer and the high level of abstraction, PCs with sufficient capacities and cloud infrastructure are used for these services. On the overall, WS implement services of the domains Multimedia, Passenger / Comfort, and Diagnostics. However, the goal of all services in the WS class is sharing information on the Internet. For the communication architecture, it is vital to provide these services with the usual communication standards of the Internet, e.g., authorization and serialization. Although the focus is on globally-used services, these services can also be used internally or in the local environment.

The *IP-based Services (IPS)* class contains all services that do not need to be implemented in any of the other classes, meaning the focus is neither on real-time capability nor on global accessibility. Accordingly, services from all domains can be implemented in this class. To be able to communicate in the local environment, standard Internet protocols are used for data transmission. These services should be able to run on almost any device.

## IV. Middleware Design

This section presents our system architecture, which is inspired by SOME/IP [5] and the architecture of DDS [20]. We adapt the protocol stack concept of Cucinotta et al. [15] to the automotive domain, and design QoS negotiations similar to Menascé et al. [17].

### A. *System Architecture*

The middleware architecture is depicted in Figure 1 and shows the components on the publisher and subscriber-side. There are three static middleware components on each host: A Local Service Manager (LSM) that manages all services on a device, a Local Service Registry (LSR) that provides a registry of all known services, and a Quality-of-Service Manager (QoSM) that implements the QoS Negotiation Protocol (QoSNP) and provides brokers to carry out the negotiations. In this work, the LSR implements a static version, as a list of all existing services. In the future, this could be extended to use the SOME/IP service discovery module. Although various communication patterns could be implemented, this work focuses on a publish/subscribe scheme since this is a very suitable pattern for in-car communication [5].

A service is realized with the following three components (see Figure 1) to achieve a separation between data and communication; the middleware dynamically provides the latter two: (1) The providing/consuming user application itself, (2) an endpoint implementing the concrete protocol and transferring/receiving data to/from the network, and (3) a connection module between the two. When an application registers a new service, the LSM returns a connector module to the application. Via the connection module, the application can transmit or receive data. When a new connection for the service is established, the middleware connects the created endpoint to the connector module of the application. To create

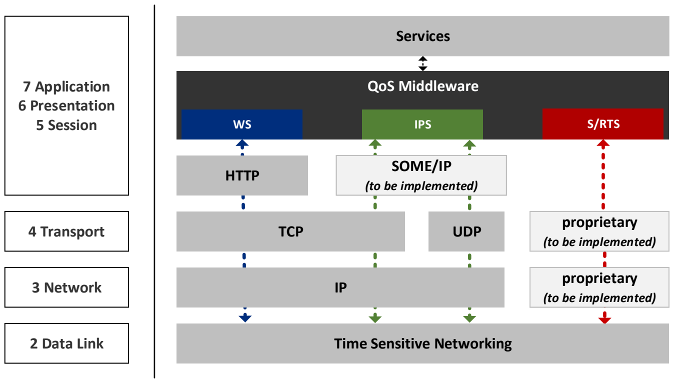

Fig. 2: Multiprotocol stack according to OSI Layers divided into service classes.

the correct connection endpoints, the QoSM implements a QoS negotiation described in Section IV-C. When the connection is set up, the data flows from the publishing application to the connector, which multiplies it for all connected endpoints. Each endpoint implements one of the described QoS classes. The publishing endpoints then send the data to all connected subscribers in the network. On the subscriber-side, a connection-specific endpoint receives and unpacks the data. Finally, the receiving endpoint forwards the data to all subscribing applications at the host through their connector module.

## B. Protocol Stack

Figure 2 displays the protocol stack used by the middleware. By means of such multi-protocol stack, the middleware can select paths through the stack according to the QoS properties. Appropriate endpoints can be created and connected at the application, while the specific protocol remains hidden from the application.

The communication of the RTS is implemented using TSN priorities. By this, streams are reserved in the network, and hard real-time guarantees are provided for previously registered and reserved data volumes. In the current version, the RTS class is transmitted directly onto layer 2 without using network or transport protocols. In a future version, UDP or SOME/IP could be equally used provided protocols are correctly mapped.

For services of the IPS class, a plain IP-based stack is used that is transferred to layer 2 with best effort. Depending on the QoS requirements, TCP or UDP is used at the transport level. In a future version, support for SOME/IP could be added.

The WS class uses HTTP on top of the TCP/IP stack. Thus, all modern web service implementations can be realized such as SOAP or ReST. The WS class is not implemented yet because we focused on an existing automotive network that does not include web services.

## C. Quality-of-Service Negotiation

To establish communication relations with service-level policies, a QoS agreement must be negotiated between the provider and the consumer. It determines the protocols that are used for transmission. On creation, a provider service specifies all requirements it can meet in the form of QoS class offers. On the other end, the consumer specifies its request requirements. Since the QoS requirements vary per client, the QoS negotiation targets the needs of the client.

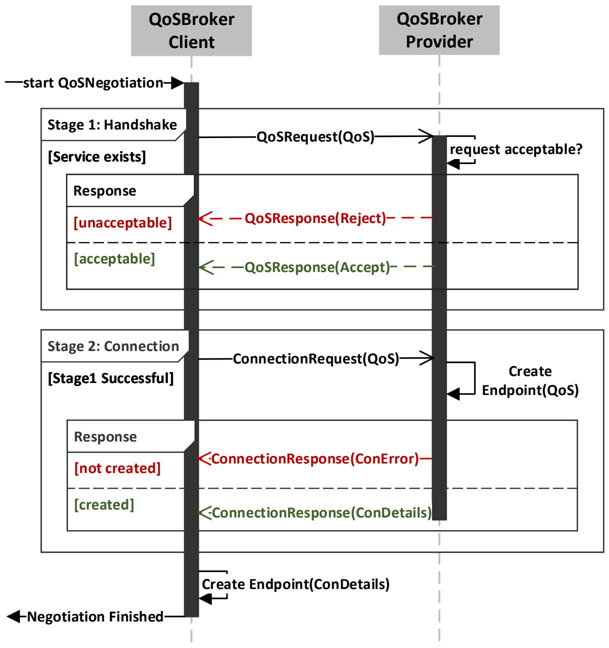

Fig. 3: Sequence diagram of the connection setup with the proposed QoS Negotiation Protocol.

On the client-side, the Quality-of-Service Manager (QoSM) creates a QoS Broker that handles the negotiation for the consumer. If a request from a QoS Broker arrives at the provider-side, its QoSM first checks whether the requested service exists on this machine. If it exists, a QoS Broker is also created on this side, representing the provider at the negotiation. Now the negotiation can begin.

The QoS Negotiation Protocol (QoSNP) is implemented by the QoS Broker as a state machine, for both the consumer and the provider-side. Figure 3 shows a sequence diagram of the connection setup with the QoSNP.

The protocol consists of two phases. The first phase performs a handshake. The consumer's broker sends a request with the required QoS properties. The broker of the provider confirms that the service is reachable at the specified address and that the executing machine of the provider would generally be able to serve the consumer's request. The latter means that it supports both the requested protocols and QoS properties, as well as having enough resources to create a new endpoint on the system. If so, then a response is sent that the request is acceptable. If the request can not be accepted, it is also communicated. At this point, a future extension could be a re-negotiation with possible *graceful degradation*, as described by Abdelzaher et al. [18]. Thus even with (partial) non-fulfillment of the requirements, data could be exchanged.

The second phase creates the endpoints for the connection according to the agreement. For this purpose, a message is sent by the consumer's broker requesting the connection with the negotiated properties. As a result, an appropriate endpoint is created on the provider-side if it does not already exist. The middleware then connects the endpoint to the provider application via its connector module. The provider's broker then sends the connection details such as transport port or stream ID to the consumer's broker, and the corresponding endpoint is created. Finally, a connection with the endpoint of the provider is established, and the data can be sent.

### D. Implementation in OMNeT++

The middleware has been implemented as a framework for the discrete event simulation OMNeT++[2] to make it accessible for evaluation. Our implementation is based on the INET framework[3], providing internet protocols and the CoRE simulation frameworks developed in previous work [26], providing real-time Ethernet protocols as well as automotive bus systems such as CAN. The implemented simulation model is available as open-source[4].

## V. Evaluation

We are now ready for investigating the performance of our middleware by a use case driven simulation using the OMNeT++ network simulator. We concentrate on the real-time capabilities, which are the most critical service guarantees in the automotive domain. Since Seyler et al. [27] already evaluated the impact of service discovery, we focus our evaluation on (1) the impact of the QoS negotiation on the setup time of all services and (2) latency ranges of different QoS classes.

Almost all ECUs of a parked vehicle are inactive before use. As soon as the driver contacts the car, e.g., via keyless entry ECUs and services are activated step by step. First, the services of the door ECU or the service "Active interior lighting" must be available very quickly. Services of the engine ECU may be available later after the driver has taken a seat in his car. In current cars the door ECU must be working after $\approx 150\,\mathrm{ms}$ to $200\,\mathrm{ms}$. Hence, negotiation times for services that must be quickly available should be at least an order of magnitude smaller. Advantageously, only a small amount of network traffic takes place in the wake-up phase of the vehicle. Two networks of $1\,\mathrm{Gbit/s}$ capacity will be investigated.

(A) The simple network visualized in Figure 4. It consists of a node that hosts several publishers, several subscriber nodes, two nodes that generate cross traffic, and two switches. To evaluate different configurations, we will vary the number of publishers on the publisher node, the number of subscriber nodes, and the number of subscribers on each subscriber node.

(B) A realistic automotive network based on a real communication matrix. This database contains anonymized information for the sender, receiver and timings of all CAN messages in a production vehicle.

[2]OMNeT++ Simulation Environment: https://omnetpp.org/
[3]INET framework: https://inet.omnetpp.org/
[4]SOQoSMW simulation model: https://github.com/CoRE-RG/SOQoSMW

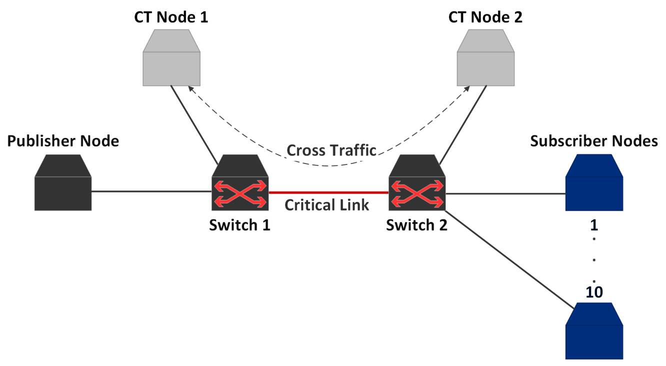

Fig. 4: (A) Simple network consisting of a publisher, varying subscribers, and cross-traffic.

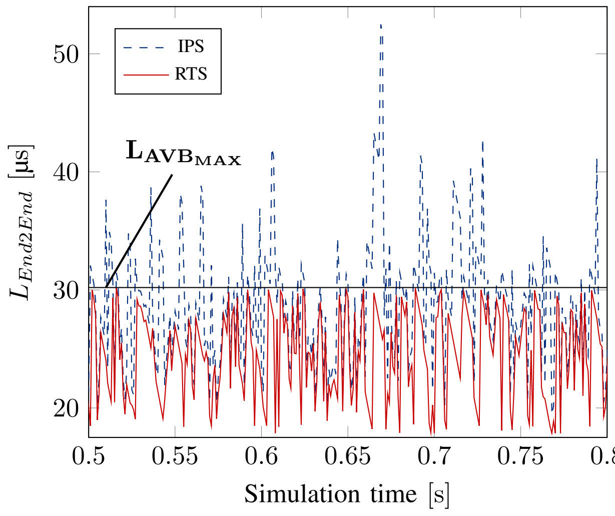

Fig. 5: End-to-end latency analysis with different service classes under varying cross traffic.

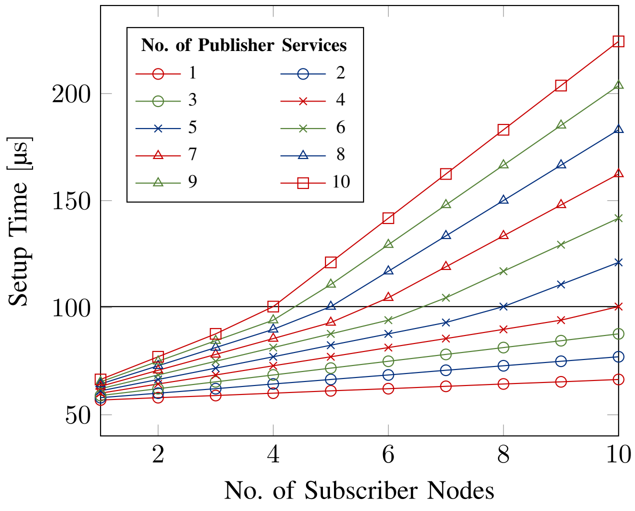

Fig. 6: Setup times for increasing numbers of subscriber nodes. In each run a new publisher service is added to which all subscribers subscribe.

### A. Simple Network Evaluation

Based on the simple network of Figure 4 we measure setup times and latencies for different QoS classes. To observe the required time of each QoS negotiation separately, the first set of simulation runs investigates one subscriber using different QoS classes without cross traffic (CT). Essentially, the setup time of a service consists of the time for QoS negotiation plus the time to establish connectivity between a publisher and subscriber. The duration of the QoS negotiation lasts about $76\,\mu s$ independent of the QoS class. For IPS, a TCP connection has been established after $130\,\mu s$ and UDP is ready after $60\,\mu s$. For RTS using AVB the connection has been established after $100\,\mu s$, while connectionless services are available right after the negotiation. The different times for TCP and AVB result from different overheads for the connection establishment.

Figure 5 shows the latency behavior of RTS and IPS with CT. $L_{\mathrm{AVB}_{max}}$ is the latency limit AVB requires for this topology. The left CT-Host sends full size frames in a normal distribution between $\approx 2\,\mu s$ and $23\,\mu s$ (resulting in a bandwidth $\approx 950\,\mathrm{Mbit/s}$). The hardware delay of the switches is $8\,\mu s$ and the processing time of the publisher and subscribers $20\,\mathrm{ns}$ each. Since only one link with CT exists in our simple network, the maximal latency must not exceed $30.2\,\mu s$. This configuration consists of three subscribers, one of which using RTS and the other two use IPS as QoS-class. In the simulation a maximum latency of $93\,\mu s$ for IPS and $30\,\mu s$ for RTS was captured in a $20\,\mathrm{s}$ run. This shows that mixing different service classes does not exceed $L_{\mathrm{AVB}_{max}}$ for RTS traffic.

Figure 6 compares the setup times as functions of the number of subscriber nodes for different publisher service multiplicities. All publisher services run on the same node, while for each publisher service one subscriber service runs on each subscriber node. The figure shows that the setup times are increasing (over-)proportionally with the number of QoS negotiations in the network. The horizontal line in the figure at the $100\,\mu s$ mark for the setup time indicates a change in the linear behavior at approximately 40 negotiations. The network becomes overloaded at this point and traffic jams delay the negotiation process. Since the services of a car will be started stepwise, this situation should not occur in a real car network.

### B. Realistic Automotive Network Evaluation

We now take a closer look at the realistic automotive network as visualized in Figure 7. This network is a variation of a legacy network of an upper middle class production car. The legacy network consists of domain-specific CAN-buses connected via a central gateway. In the legacy network every ECU of the same domain is connected to the CAN-bus of this domain regardless of its location in the car. In our variation these CAN-buses are split up into sub-buses according to nine spatial zones distributed across the vehicle. A CAN-to-Ethernet gateway connects each sub-bus to a switched Ethernet backbone. Within Figure 7, each domain-specific CAN bus is represented by its own color. The numbers marking the outer edges of each sub-bus denote the number of CAN nodes connected to this sub-bus.

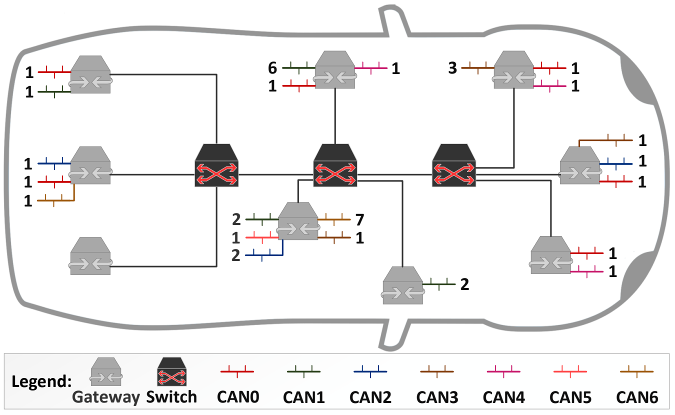

Fig. 7: (B) Realistic automotive network based on real in-car communications.

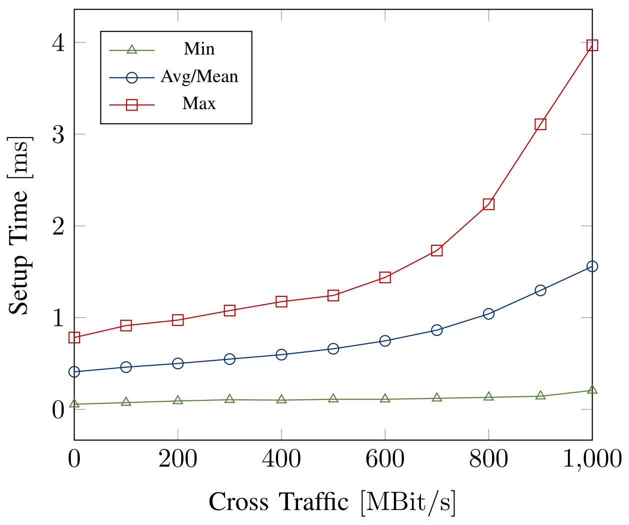

Fig. 8: Minimum, maximum, and average setup time in a realistic network with CT.

A Corresponding gateway receives the CAN messages an ECU generates. The gateway then acts as a publisher of this information and provides a service for each CAN message. All gateways connected to CAN ECUs needing the information of such a CAN message will subscribe to those services. If a subscribing service at a gateway receives Ethernet messages, it forwards them to the correct CAN-bus.

Figure 8 depicts the growths of the setup times with increasing CT. All links in the network will be loaded with CT from $0\,\mathrm{Mbit}$ until $1000\,\mathrm{Mbit}$. For every simulation, we captured the minimum, average, and maximum setup time. There are only slight changes in the minimum times, while the average and maximum times seem to rise almost exponentially. At a link load of about $60\,\%$ CT, the increase in setup time is drastic enough that negotiations might not finish in time. High volumes of CT above a certain threshold will lead to packet loss, which we did not account for in our simulations.

However, the results of Figure 8 show protocol delays of a few milliseconds. These results clearly comply with the requirements of approximately $150\,\mathrm{ms}$ to $200\,\mathrm{ms}$ as discussed at the beginning of this section.

## VI. Conclusion and Future Work

In this paper, we presented a service-oriented communication middleware tailored to the requirements of the car. This approach can meet heterogeneous requirements using a variable protocol stack and a QoS Negotiation Protocol, which allows services to dynamically negotiate their demands. This way QoS guarantees can be provided to clients based on their QoS class. We evaluated our concept following a case study in simulations based on a realistic automotive network. We could show that the middleware and its dynamic QoS negotiations successfully support mixed communication networks of heterogeneous requirements. By using a multi-protocol stack, variable QoS regimes can interoperate while preserving the flexibility of service-oriented communication. We analyzed the impact of the negotiation on the setup time of the network. Our findings indicate that up to 70 % cross-traffic are compliant with a maximum of $2\,\mathrm{ms}$ for the setup, which is clearly acceptable for most of the traffic of the automotive network. For safety-critical traffic that is particularly susceptible to jitter, we recommend the use of statically defined TDMA classes, which experience no delay.

In future work, we will build a demonstrator of a real car components to determine real-world runtime delays and to analyze the various interactions within such a system.

## References

[1] C. Buckl, A. Camek, G. Kainz, C. Simon, L. Mercep, H. Stähle, and A. Knoll, "The Software Car: Building ICT Architectures for Future Electric Vehicles," in *2012 IEEE International Electric Vehicle Conference*, Mar. 2012, pp. 1–8.

[2] fortiss GmbH, "The Software Car: Information and Communication Technology (ICT) as an Engine for the Electromobility of the Future," fortiss GmbH, Tech. Rep., Mar. 2011, summary of results of the "eCar ICT System Architecture for Electromobility" research project sponsored by the Federal Ministry of Economics and Technology.

[3] K. Matheus and T. Königseder, *Automotive Ethernet*. Cambridge, United Kingdom: Cambridge University Press, Jan. 2015.

[4] Institute of Electrical and Electronics Engineers, "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pp. 1–1993, Jul. 2018.

[5] AUTOSAR, "SOME/IP Protocol Specification," AUTOSAR, AUTOSAR Standard 696, Nov. 2016.

[6] P. Meyer, F. Korf, T. Steinbach, and T. C. Schmidt, "Simulation of mixed critical in-vehicular networks," in *Recent Advances in Network Simulation*. Springer, 2019, pp. 317–345.

[7] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)," in *2012 IEEE Vehicular Technology Conference (VTC Fall)*. Piscataway, NJ, USA: IEEE Press, Sep. 2012.

[8] G. L. Gopu, K. V. Kavitha, and J. Joy, "Service Oriented Architecture based connectivity of automotive ECUs," in *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, Mar. 2016, pp. 1–4.

[9] H.-T. Lim, L. Völker, and D. Herrscher, "Challenges in a future IP/Ethernet-based in-car network for real-time applications," in *48th ACM/EDAC/IEEE Design Automation Conference (DAC), 2011*, Jun. 2011, pp. 7–12.

[10] A. Scholz, I. Gaponova, S. Sommer, A. Kemper, A. Knoll, C. Buckl, J. Heuer, and A. Schmitt, "$\epsilon$SOA - Service Oriented Architectures adapted for embedded networks," in *2009 7th IEEE International Conference on Industrial Informatics*, Jun. 2009, pp. 599–605.

[11] J. Hill, H. Sutherland, P. Stodinger, T. Silveria, D. C. Schmidt, J. Slaby, and N. Visnevski, "OASIS: A Service-Oriented Architecture for Dynamic Instrumentation of Enterprise Distributed Real-Time and Embedded Systems," in *2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, May 2010, pp. 10–17.

[12] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Zirkler, L. Fiege, M. Armbruster, G. Spiegelberg, and A. Knoll, "RACE: A Centralized Platform Computer Based Architecture for Automotive Applications," in *2013 IEEE International Electric Vehicle Conference (IEVC)*, Oct. 2013, pp. 1–6.

[13] M. G. Valls, I. R. Lopez, and L. F. Villar, "iLAND: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 228–236, Feb. 2013.

[14] M. Wagner, D. Zöbel, and A. Meroth, "SODA: Service-Oriented Architecture for Runtime Adaptive Driver Assistance Systems," in *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, Jun. 2014, pp. 150–157.

[15] T. Cucinotta, A. Mancina, G. F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A Real-Time Service-Oriented Architecture for Industrial Automation," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 3, pp. 267–277, Aug. 2009.

[16] F. Jammes and H. Smit, "Service-Oriented Paradigms in Industrial Automation," *IEEE Transactions on Industrial Informatics*, vol. 1, no. 1, pp. 62–70, Feb. 2005.

[17] D. Menascé, H. Ruan, and H. Gomaa, "QoS management in service-oriented architectures," vol. 64, pp. 646–663, Aug. 2007.

[18] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "QoS Negotiation in Real-Time Systemsand Its Application to Automated Flight Control," *IEEE Transactions on Computers*, vol. 49, no. 11, pp. 1170–1183, Nov. 2000.

[19] M. Becker, Z. Lu, and D.-J. Chen, "Towards QoS-Aware Service-Oriented Communication in E/E Automotive Architectures," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 4096–4101.

[20] Object Management Group, "Data Distribution Service," Online, OMG, Standard DDS 1.4, Mar. 2015, access: 2018-02-08.

[21] M. Broy, *Model-Driven Development of Reliable Automotive Services - Second Automotive Software Workshop, ASWSD 2006, San Diego, CA, USA, March 15-17, 2006, Revised Selected Papers*, 2008th ed. Berlin Heidelberg: Springer Science & Business Media, 2008.

[22] J. Schäuffele and T. Zurawka, *Automotive Software Engineering*. Wiesbaden: Vieweg und Teubner, 2010.

[23] A. Pretschner, M. Broy, I. H. Krüger, and T. Stauner, "Software Engineering for Automotive Systems: A Roadmap," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, May 2007, pp. 55–71.

[24] M. E. Jobst and C. Prehofer, "Towards Hierarchical Information Architectures in Automotive Systems," in *2016 3rd International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC)*, Apr. 2016, pp. 41–46.

[25] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Beware of the Hidden! How Cross-traffic Affects Quality Assurances of Competing Real-time Ethernet Standards for In-Car Communication," in *2015 IEEE Conference on Local Computer Networks (LCN)*, Oct. 2015, pp. 1–9, lCN Best Paper Award.

[26] T. Steinbach, H. Dieumo Kenfack, F. Korf, and T. C. Schmidt, "An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy," in *SIMUTools 2011 – 4th International OMNeT++ Workshop*. New York, USA: ACM DL, March 21-25 2011, pp. 375–382.

[27] J. R. Seyler, T. Streichert, M. Glaß, N. Navet, and J. Teich, "Formal Analysis of the Startup Delay of SOME/IP Service Discovery," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 49–54.