Quick SOME/IP – QUIC-based Service-Oriented Middleware for Software-Defined Vehicles

Yang Wu Chien-Chung Shen
Department of Computer and Information Sciences
University of Delaware, USA
wuyang,cshen@udel.edu

2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall) | 979-8-3315-1778-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/VTC2024-Fall63153.2024.10757787

***Abstract*—Spurred by technological advancements and consumer demands, the automotive industry is moving towards software-defined vehicles. To accommodate the complexities of software that execute on a system of networked ECUs (Electronic Control Units) and sensors, service-oriented architecture (SOA) becomes a viable architecture. SOME/IP (Scalable Service-Oriented MiddlewarE over IP) is a middleware standardized by AUTOSAR that implements SOA for automotive systems. By leveraging the unique features of the QUIC (Quick UDP Internet Connections) transport protocol, we design "Quick" SOME/IP by substituting the TCP used in SOME/IP with QUIC. In addition to supporting the four communications patterns offered by SOME/IP, Quick SOME/IP introduces fault tolerance streaming as the fifth communication pattern, which facilitates simultaneous data transmissions from one sender to one receiver over two physically disjoint paths to tolerate the failure of one path. Using the Nexus open-source C++ library implementation of QUIC, we modified vsomeip, an open-source C++ implementation of SOME/IP using TCP, to implement qsomeip. Our experiments demonstrated that qsomeip outperforms vsomeip in terms of reduced connection latency and enhanced security with the new capability of fault tolerant streaming.**

***Index Terms*—SOME/IP, QUIC, Service-oriented Architecture, Software-defined vehicle**

I. Introduction

Software-defined vehicle (SDV) describes a vehicle whose features, capabilities, and performance are primarily governed by software [1]. The shift towards SDV is driven by advancements in computing power, connectivity, and software capabilities, and by changing consumer expectations. By incorporating software at the core of vehicle design, manufacturers can leverage these technological advancements to offer new features and improve performance. For instance, software can enhance vehicle safety via advanced driver-assistance systems (ADAS), including features like automatic braking, lane-keeping assist, and adaptive cruise control. These features can be improved and updated via Over-the-Air (OTA) software updates [2], much like updating the operating system or apps on smartphones, to reduce the costs associated with physical recalls and mechanical upgrades. The ultimate goal of SDV is to develop fully autonomous vehicles, which rely on software to control all aspects of driving and allow vehicles to become more than just means of transportation [3].

Service-oriented architecture (SOA) [4] is a software design paradigm where applications are structured as a collection of loosely coupled, independent services. These services communicate through a protocol over a network with each other to perform complex processes. SOA aims to improve software systems' agility, efficiency, and maintainability by promoting modularity and reuse. In the context of SDV, SOA allows for the creation of modular applications that can be developed, deployed, and updated independently [5], which is crucial for SDV as different vehicle functionalities (such as infotainment, autonomous driving, and telematics) can be deployed and updated individually without affecting other systems. Ultimately, SOA supports the visions of SDV that are continually improving, highly customizable, and capable of integrating with the broader ecosystem of transportation and smart city technologies.

SOME/IP (Scalable service-Oriented MiddlewarE over IP) [6] is a middleware standardized by AUTOSAR (AUTomotive Open System ARchitecture) that implements SOA for automotive systems, where distinct software components provide SDV above functionalities. SOME/IP is structured around sending messages over IP networks, utilizing either TCP or UDP based on the application's needs, following the four communication patterns of *request/response*, *fire/forget*, *event*, and *field*. Vsomeip [7] is an open-source implementation of the communication and service discovery functions of SOME/IP.

QUIC (Quick UDP Internet Connections) [8] is an IETF-standardized transport layer protocol that combines the best features of TCP and UDP while mitigating their respective drawbacks. QUIC offers UDP's speed and simplicity with TCP's reliability and sequencing, enriched with security and performance enhancements. For instance, QUIC combines the connection establishment and the transport security negotiations into a single step, thereby reducing latency, especially for new connections over previously established sessions (termed 0-RTT handshake). In addition, QUIC incorporates TLS as a core protocol part, not an add-on, so that security becomes a fundamental aspect of QUIC, providing better privacy and security guarantees compared to TCP, which relies on TLS/SSL to be layered on top separately.

In this paper, we substitute QUIC for TCP in SOME/IP to design "Qucik" SOME/IP and to prototype qsomeip and compare its performance with vsomeip. The qsomeip is expected to incur lower connection delay and provide more robust security. In addition, by utilizing QUIC's Connection Identifiers (CIDs) and stream multiplexing, we also introduce the fifth communication pattern, termed *fault tolerance streaming* to facilitate simultaneous data transmissions from one sender to

one receiver over two physically disjoint paths to tolerate the failure of one path. Through rigorous experimentation, we show that qsomeip outperforms vsomeip in terms of connection setup latency, security, and fault tolerance for in-vehicle networking systems.

The remainder of the paper is organized as follows. The next section reviews SOME/IP, QUIC, and vsomeip. Section III describes the prototype of "Quick" SOME/IP, qsomeip. Section IV evaluates the performance of qsomeip and compares it with vsomeip. Section V summarizes the contributions of this work.

II. Background

A. Review of SOME/IP

SOME/IP is a middleware standard that supports SOA for on-board vehicular networks. SOME/IP specifies the API for applications to communicate through TCP or UDP over IP. Fig. 1 depicts the four communication patterns supported by SOME/IP: request/response, fire/forget, event (subscribe/notify), and field. The request/response pattern realizes remote-procedure call (RPC). When a response is not required for a request, we have the fire/forget pattern. The event pattern is used to report the status to interested parties, which is realized in terms of subscription and notification. In the field pattern, a field is a property of a service that can be remotely accessed using getters or setters. Getter is the method to read field value; setter is the method to set the field value. When a field's value changes, a notification event is sent out by the notifier.

B. Review of QUIC

QUIC is a protocol developed by Google that operates on top of UDP, as depicted in Fig. 2. Compared to TCP, QUIC offers numerous advantages, such as zero-round-trip time (0-RTT) connections, enhanced congestion control, connection migration, and forward error correction. The 0-RTT connection is the most significant performance advantage of QUIC [9]. In most circumstances, just 0-RTT is required to perform data transfer, which is crucial for autonomous driving systems that make decisions by transmitting data with minimal latency. Meanwhile, QUIC employs CIDs to decouple connection identification from specific IP addresses and ports. This architectural choice enables seamless migration of connections across different network paths without interruption. Consequently, QUIC possesses the inherent potential for supporting multipath capabilities.

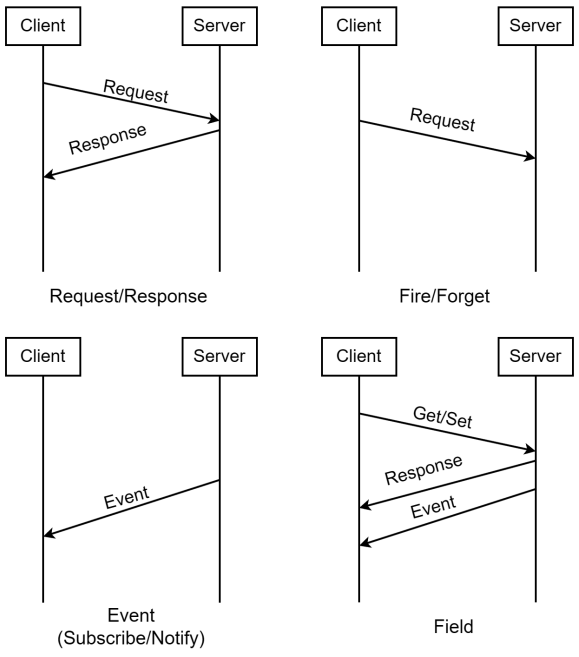

Fig. 1. Communication patterns of SOME/IP

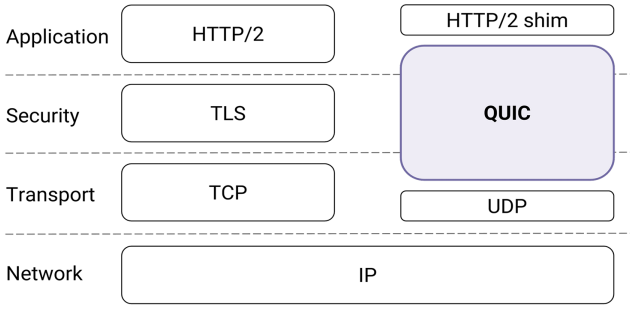

Fig. 2. Comparison between TCP and QUIC in protocol stack [8]

C. Review of vsomeip Architecture

Vsomeip is an open-source implementation of the communication and service discovery functions of SOME/IP. Fig. 3 depicts the software architecture of vsomeip [6] with two ECUs connected by an Ethernet link. The routing manager is the core of the vsomeip and is responsible for local and remote communications. It manages transport endpoints (i.e., TCP and UDP sockets) to communicate with applications residing on remote devices and local endpoints (i.e., Unix domain sockets) with other applications residing on the same device.

III. Prototype of Quick SOME/IP (qsomeip)

Boost.Asio is a cross-platform C++ library for network and low-level I/O programming that provides a consistent asynchronous model. Due to vsomeip's use of the Boost.Asio C++ library's `boost:asio::io_service` class [6], we chose Nexus [10] to prototype qsomeip, as Nexus uses Boost.Asio's `boost::asio::any_io_executor` class, which is constructed using the executor obtained from the `boost:asio::io_service`. After modifying vsomeip to work with Nexus, we build qsomeip, which uses QUIC to facilitate the four communication patterns of SOME/IP. As the vsomeip routing manager in each device manages the transport endpoints, the routing manager of qsomeip binds to the Nexus' implementation of QUIC as specified by applications. The qsomeip will still implement the same API as the vsomeip because the routing manager has the same management logic for both TCP and QUIC nodes.

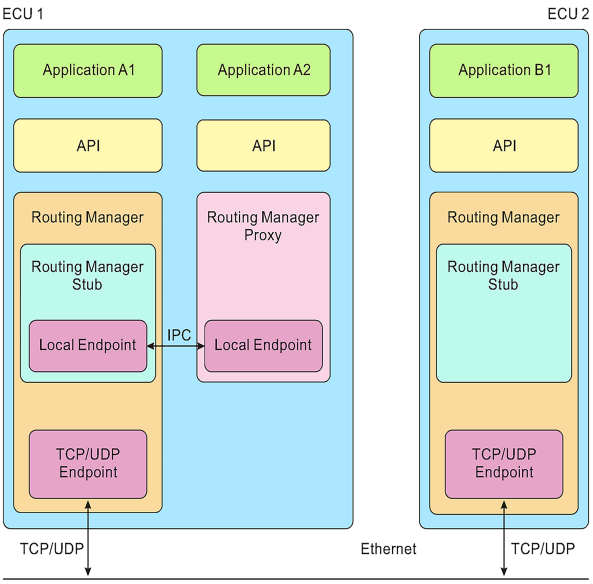

Fig. 3. Software architecture of vsomeip [6]

For transport layer security, OpenSSL is utilized by vsomeip, while Nexus employs BoringSSL. To address the compatibility issue so that OpenSSL and BoringSSL may coexist in the same process, BoringSSL is compiled as a Git submodule alongside vsomeip per BoringSSL's official build guide [11].

Two-path Fault Tolerance Streaming

As shown in Fig. 4, to provide fault tolerant transmissions of (streaming) data from vehicle sensors (e.g., camera, lidar, radar, etc.) to ECUs (Electronic Control Units), "Quick SOME/IP" defines the *fifth* communication pattern of fault tolerance streaming, to facilitate data streaming from a sender over two disjoint paths to a receiver. This communication pattern allows the same data to be transmitted *simultaneously* over two physically disjoint paths to the receiver to provide fault tolerance to data transmission in the event of connection failure, due to collision, for instance.

There exist multiple designs of multi-path extensions for QUIC [12]. We have taken inspiration from Alibaba's xquic [13] design, which introduces a multiple-port monitoring mechanism where each worker is assigned a unique port and a port's information is embedded in the CID of the returned packet. This approach enables efficient path identification and connection management without modifying the kernel.

We tested qsomeip's two-path design between one sender-receiver pair by simultaneously transmitting a video through two disjoint paths. The experiments demonstrated that when one path is cut off, the receiver continues receiving data over the other path without any interruption.

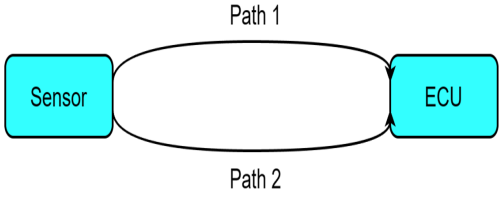

Fig. 4. Two-path Fault Tolerance Streaming Pattern

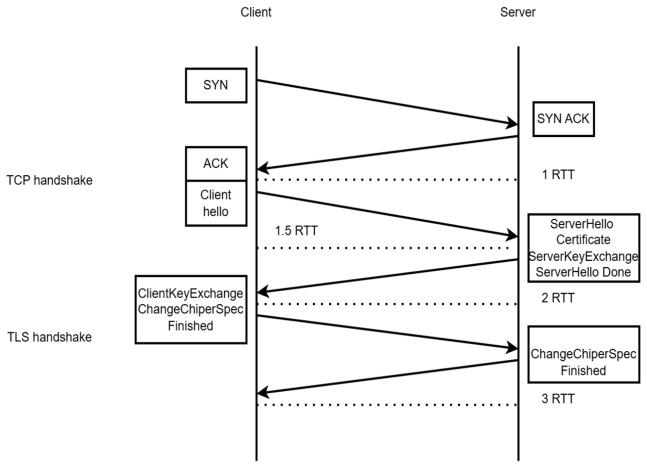

Fig. 5. Process of vsomeip connection establishment with 3-RTT

IV. Performance Evaluation

We compared the performance of qsomeip with vsomeip.

A. Lower latency for connection establishment

Compared to qsomeip, it takes vosmeip 3-RTT to establish a connection. Fig. 5 illustrates that TCP and TLS establish their respective connection states in sequence, resulting in 3-RTT between the client and server for handshake and encryption negotiation. This process is not suitable for ADAS systems that require real-time data transmission. By implementing the qsomeip, latency can be reduced.

The Diffie-Hellman (DH) algorithm is a key exchange protocol and can realize the secure establishment of the keys in network communication. Because in many cases, no matter what method is used for communication, as long as the channel is insecure, it is difficult to ensure that the data is protected. So, the QUIC protocol uses the DH algorithm to ensure data exchange security and combines its encryption and handshake process to reduce the number of round trips during the connection establishment process.

For the first connection of the QUIC protocol, its main content is the key negotiation and data transmission between the client and the server [8]. The steps for establishing a handshake in QUIC are shown in Fig. 6. It can be seen that when the first connection is made, it takes 1-RTT. The client keeps the server configuration, allowing subsequent connections to utilize the saved server configuration to bypass key negotiation directly, so the following connections require a 0-RTT handshake.

To show the performance difference between qsomeip and vsomeip, we designed experiments to measure the overall time required by qsomeip and vsomeip to complete a specified number of request tasks under the same configuration. Since QUIC incurs different connection time for 1-RTT and 0-RTT

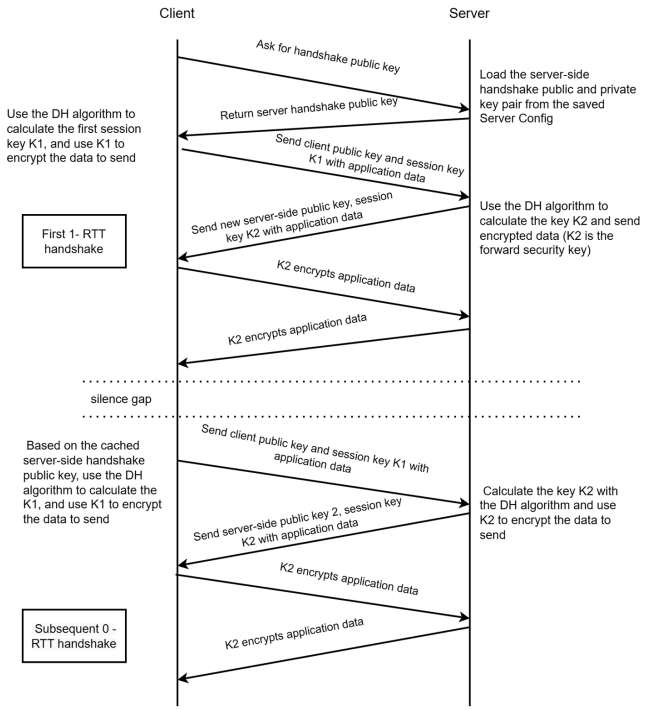

Fig. 6. Steps for establishing a connection with QUIC

connections, we will measure the overall time to complete a specified number of requested tasks in the same configuration to show the different connection establishment time. For the 1-RTT qsomeip experiments, the server config is flushed after each connection establishment. In contrast, for the 0-RTT qsomeip experiments, the first request incurs 1-RTT while the remaining requests incur 0-RTT.

We measured the latency of the request/response and the event (subscribe/notify) patterns, as the fire/forget and the field patterns are special forms of the previous two patterns. The field pattern should have the same connection time as the event pattern, and the main difference between them is the message content, which has no significance in the measured connection time. Also, because the message is ignored, there is no way to send it back and forth in the fire-forget pattern, so it is unsuitable for testing the response speed between the server and the client. We compared qsomeip (0-RTT), qsomeip (1-RTT), and vsomeip under two identically configured virtual machines for executing the measurement samples. To get accurate results, we executed three times for each target time and took the average.

For the request/response pattern, we wrote response-measurement and request-measurement samples for testing; the former was similar to a standard server that will respond directly to client requests; the latter will determine the total number of times to perform the request-response cycle based on the received command line parameters. We can assume that the server will be started first each time so that the server will remain listening, and the client will establish a connection and send it directly each time it starts. For each experiment, the client repeatedly makes a request and waits for a response for a certain number of iterations maintained by a counter. We vary the counter from 10 to 100 with increments of 10. We measured the time it takes for each experiment via the Unix `gettimeofday` function.

For the event pattern, we set notification measurement to continuously send event messages of a fixed length. Then, we added a counter to subscribe-measurement to output the program runtime and terminate it when the target number of messages was received. We put the target number for this pattern from 100 to 200 with increments of 10.

Experiment Results. As illustrated in Fig. 7 and 8, qsomeip demonstrates superior performance at the transport layer in both communication patterns. This is primarily evidenced by the reduced overall time required for qsomeip to complete a specified number of request tasks under identical configurations compared to vsomeip. Specifically, for both 0-RTT and 1-RTT in qsomeip, the 0-RTT configuration exhibits a lower overall time to complete the specified number of request tasks. This is because 0-RTT allows the handshake between client and server to be completed faster to establish the connection. However, it is noteworthy that qsomeip's performance advantage diminishes as the number of requests increases. With the extension of the overall communication duration, the performance of qsomeip and vsomeip gradually converges. This trend is particularly pronounced in the event pattern, which is not obviously in the request/response pattern. This is because neither the server nor the client waits for each other during the event pattern's data transmission, and the server transmits data at a high frequency unless it encounters self-imposed blocking.

Based on the experimental results, qsomeip is well-suited for Advanced Driver Assistance Systems (ADAS) due to its operation within a communication environment characterized by very low and stable latency.

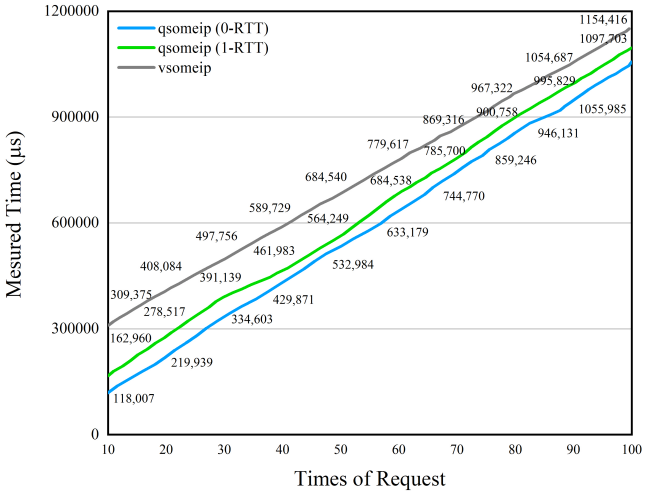

Fig. 7. Experimental results of request-response pattern

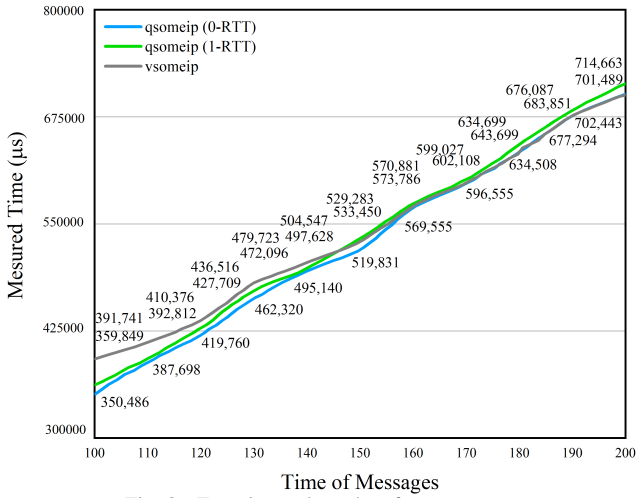

Fig. 8. Experimental results of event pattern

B. Stronger security protection for communications

In the context of security, the most significant difference between qsomeip and other security improvements of SOME/IP is that qsomeip provides more robust security protection, as offered by TLS 1.3, while applying the QUIC protocol and not adding extra communication delays. In some existing designs, they tried to build a security protocol with authentication and confidentiality tests to protect the middleware, which improved security with increased latency [6]. Specifically, according to their experimental results, the 1-RTT latency increased by applying local communication to transmit the message to the remote host, and packets flow over the Internet. Opposite of this, our result was that the qsomeip's 1-RTT connection establishment delay was reduced compared to the vsomeip's 3-RTT connection establishment delay. Since qsomeip uses QUIC to replace TCP, the security of the middleware is enhanced. For instance, QUIC supports forward security when a connection is established [8]. Forward security signifies that previously encrypted data will not be compromised even if the key itself is compromised.

To demonstrate the security feature of qsomeip, we tested qsomeip to defend against the man-in-the-middle (MITM) attacks that cause damages to the vsomeip implementation of SOME/IP [14]. In the MITM attack, the Service offering server first transmits a service offer message with its endpoint. When receiving this message, the Attacker, as the middleman, will send the same service offer with its endpoint to the client. Not being able to differentiate the two service offers, the client may choose the attacker as the service provider for its requests. If the attacker (middleman) is chosen, the attack becomes successful.

By using the DH algorithm in TLS 1.3 within QUIC, all data transmitted over QUIC is encrypted with private keys. This ensures that even if an attacker intercepts the data, she cannot read the contents without the private keys. Also, during the TLS handshake, the server presents a digital certificate that the client verifies. This certificate is signed by a trusted third party, ensuring the client communicates with the legitimate server, not an attacker. Our experiments validated that man-in-the-middle attacks cannot succeed.

V. Conclusion

This paper presents "Quick" SOME/IP, which replaces TCP with QUIC for SOME/IP, and its implementation qsomeip. In addition, "Quick" SOME/IP introduced fault tolerance streaming as the fifth communication pattern to facilitate fault tolerant data transmission between a sender and a receiver. Based on our evaluation, qsomeip outperforms vsomeip in terms of lower connection latency, better security without extra delay, and fault tolerance data transmission for in-vehicle systems. "Quick" SOME/IP provides a potential alternative to SOME/IP for the automotive industry.

References

[1] S. Lu and W. Shi, "The emergence of vehicle computing," *IEEE Internet Computing*, vol. 25, no. 3, pp. 18–22, 2021.

[2] Y. Qi, G. Yang, L. Liu, J. Fan, A. Orlandi, H. Kong, W. Yu, and Z. Yang, "5g over-the-air measurement challenges: Overview," *IEEE Transactions on Electromagnetic Compatibility*, vol. 59, no. 6, pp. 1661–1670, 2017.

[3] S. Han, D. Cao, L. Li, L. Li, S. E. Li, N.-N. Zheng, and F.-Y. Wang, "From software-defined vehicles to self-driving vehicles: A report on cpss-based parallel driving," *IEEE Intelligent Transportation Systems Magazine*, vol. 11, no. 1, pp. 6–14, 2019.

[4] M. Rumez, D. Grimm, R. Kriesten, and E. Sax, "An overview of automotive service-oriented architectures and implications for security countermeasures," *IEEE Access*, vol. 8, pp. 221 852–221 870, 2020.

[5] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2021.

[6] M. Iorio, A. Buttiglieri, M. Reineri, F. Risso, R. Sisto, and F. Valenza, "Protecting in-vehicle services: Security-enabled SOME/IP middleware," *IEEE Vehicular Technology Magazine*, vol. 15, no. 3, pp. 77–85, 2020.

[7] A. Ioana and A. Korodi, "Vsomeip - opc ua gateway solution for the automotive industry," in *2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, 2019, pp. 1–6.

[8] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The QUIC transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 183–196. [Online]. Available: https://doi.org/10.1145/3098822.3098842

[9] F. Chiariotti, A. A. Deshpande, M. Giordani, K. Antonakoglou, T. Mahmoodi, and A. Zanella, "Quic-est: A QUIC-enabled scheduling and transmission scheme to maximize voi with correlated data flows," *IEEE Communications Magazine*, vol. 59, no. 4, pp. 30–36, 2021.

[10] C. Bodley, "nexus," 2021, gitHub repository. [Online]. Available: https://github.com/cbodley/nexus

[11] D. Benjamin, "boringssl," 2014, gitHub repository. [Online]. Available: https://github.com/google/boringssl

[12] T. Viernickel, A. Froemmgen, A. Rizk, B. Koldehofe, and R. Steinmetz, "Multipath QUIC: A deployable multipath transport protocol," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–7.

[13] W. Zhao, "xquic," 2021, gitHub repository. [Online]. Available: https://github.com/alibaba/xquic

[14] D. Zelle, T. Lauser, D. Kern, and C. Krauß, "Analyzing and securing SOME/IP automotive services with formal and practical methods," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, ser. ARES '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3465481.3465748