Secure Time-Sensitive Software-Defined Networking in Vehicles

Timo Häckel, Philipp Meyer, Franz Korf, and Thomas C. Schmidt, *Member, IEEE*

***Abstract*—Current designs of future In-Vehicle Networks (IVN) prepare for switched Ethernet backbones, which can host advanced LAN technologies such as IEEE Time-Sensitive Networking (TSN) and Software-Defined Networking (SDN). In this paper, we present an integrated Time-Sensitive Software-Defined Networking (TSSDN) architecture that simultaneously enables control of synchronous and asynchronous real-time and best-effort communication for all IVN traffic classes. Despite the central SDN controller, we can validate that control can operate without a delay penalty for TSN traffic, provided protocols are properly mapped. We demonstrate how TSSDN adaptably and reliably enhances network security for in-vehicle communication. A systematic investigation of the possible control flow integrations with switched Ether-networks reveals that these strategies allow for shaping the attack surface of a software-defined IVN. We discuss embeddings of control flow identifiers on different layers, covering the range from a fully exposed mapping to deep encapsulation. We experimentally evaluate these strategies in a production vehicle, which we map to a modern Ethernet topology. Our findings indicate that visibility of automotive control flows on lower network layers enables isolation and access control throughout the network infrastructure. Such a TSSDN backbone can establish and survey trust zones within the IVN and reduce the attack surface of connected cars in various attack scenarios.**

***Index Terms*—Automotive Ethernet, IVN, Security, TSN, SDN**

I. Introduction

VEHICLES continuously implement new features based on sensors and actuators connected with Electronic Control Units (ECUs). Traditionally, the In-Vehicle Network (IVN) is organized in functional domains using a combination of bus systems. Automotive Ethernet has emerged as the next high-bandwidth communication technology [1]. Future IVNs will migrate to switched Ethernet [2] as shared backbones for different domains and service requirements. For time-constraint traffic, the standards of Time-Sensitive Networking (TSN) (IEEE 802.1Q [3]) add real-time capabilities to Ethernet.

Advanced Driver Assistance Systems (ADAS) and similar features increase cross-domain communication and functions implemented in software following a Service-Oriented Architecture (SOA). These services are highly dynamic and require an adaptable IVN. Software-Defined Networking (SDN) [4] has been identified as a powerful building block for IVNs, as it promises to increase robustness and adaptability [5]–[7]. In SDN, the control plane of the network devices is offloaded to a central controller. On the data plane, network devices forward packets based on pipelines controlled by the SDN controller. Time-Sensitive Software-Defined Networking (TSSDN) was introduced to enable centralized reconfiguration of time-sensitive communication [8]. In recent work, we integrated TSN with SDN to control asynchronous real-time traffic using the OpenFlow protocol without a delay penalty [6].

Security challenges arise from communication with other vehicles or roadside units (V2X), and via Internet uplinks that open a vehicle to remote attackers. All this, and also the flattening interconnect of domains increase the vulnerability of safety-critical functions and require versatile measures to secure future vehicles [9]. Current vehicles are vulnerable to manipulation by third parties, which became apparent through cyber-attacks in the field [10]. A robust IVN can limit the attack surface and reduce the impact on communication.

SDN can improve network security by separating the control from the data plane [11]. The central controller has global network knowledge and devices only forward packets according to predefined flows. This bears the potential to detect, prevent, or mitigate cyber-attacks [12]. In previous work, we identified the SDN matching pipeline as a powerful tool to accurately identify, separate, and protect native IVN communication [13].

This paper develops TSSDN further by integrating transactional updates for synchronous real-time traffic and thereby completes the set of fully programmable options for controlling all classes of in-vehicle communication via a central SDN controller. Applying control programming to critical real-time traffic significantly extends network security mechanisms for in-vehicle communication. Our main contributions read:

1) We integrate TSN with SDN in a switching architecture that enables central monitoring and control for all classes of IVN communication without delaying real-time traffic.
2) We present and evaluate an approach to secure in-vehicle TSSDN by introducing reliable static configuration and secure adaptive communication.
3) We evaluate different approaches to embed in-vehicle control flows in SDN and quantify the precision with which the network can identify and isolate them.
4) We show that TSSDN can isolate in-vehicle control flows in a shared environment, prevent unwanted traffic, and significantly reduce the attack surface in a prototype built from a production vehicle.

The remainder of this work is organized as follows. Section II reviews the IVN, TSN, and SDN together with related work on network security in cars. Section III introduces the concept of TSSDN. SDN-enabled security measures for IVNs are presented in Section IV. Section V validates the TSSDN

The authors are with the Department of Computer Science, Hamburg University of Applied Sciences (HAW), 20099 Hamburg, Germany (e-mail: {timo.haeckel, philipp.meyer, franz.korf, t.schmidt}@haw-hamburg.de).

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

architecture in simulations. Section VI analyses the security impact of our access-control in a prototype car and investigates benefits and limits in attack scenarios. Finally, Section VII concludes our paper with an outlook on future work.

II. Background and Related Work

A. Evolution of In-Vehicle Networks

Modern IVNs connect sensors, actuators, and ECUs. Some in-vehicle communication is static and well-defined, such as motor control, while other is dynamic and not always active, such as ADAS. All communication relations between the installed ECUs are specified in a communication matrix. Each control flow in the matrix has exactly one sender, a list of receivers, and a unique identifier across the vehicle.

1) Traffic types: In-vehicle communications include periodic control messages, sporadic events, and media traffic. Besides the vehicle-specific traffic, there is also network control, which is required for reconfiguration and service discovery, and Best-Effort (BE) communication.

The different traffic classes have varying Quality-of-Service (QoS) requirements. Control communication often uses small frames in control loops with a fast period and may be very susceptible to jitter which requires fixed latencies around 100 µs. Media traffic (of e.g., LIDAR systems) may require several Gbit/s with guaranteed latencies around 10 ms. Safety-critical messages may have no tolerance for packet loss and require redundancy and hard deadlines. For some applications, retransmission may compensate for packet loss, while with cyclic control communication, retransmission is irrelevant.

2) Network topology: IVN topologies have evolved to adapt to the increasing communication demands of automotive applications [1], [2], [9], [14]. Initially (Figure 1a), ECUs were grouped into functional domains such as chassis control, powertrain, comfort, and infotainment using heterogeneous bus systems such as Controller Area Network (CAN), Media Oriented System Transport (MOST) and Local Interconnect Network (LIN), which physically extended over large areas of the vehicle. For cross-domain functions, a central gateway was installed to transfer messages from one domain to another.

Today, cross-domain communication is increasingly required to enable features such as ADAS, and new sensors such as high-resolution cameras need high bandwidths for communication. Figure 1b shows a domain controller topology which uses a switched Automotive Ethernet [1] backbone that enables fast cross-domain communication. Domain gateways integrate legacy devices and forward messages between the domain buses and the Ethernet backbone. Central compute units were introduced as 'High-Performance Computers (HPCs)'. They offer much higher processing power than traditional ECUs and bundle virtual functions of computationally intensive tasks.

Future vehicles will communicate with other vehicles, roadside units (V2X) and the Internet. More and more functions will be implemented in software and the number of OEM model variants will increase. This requires higher flexibility of software components and increases the dynamics of the network. The zone model (1c) connects all ECUs with a zone controller in physical proximity (e.g., front left), so that wiring

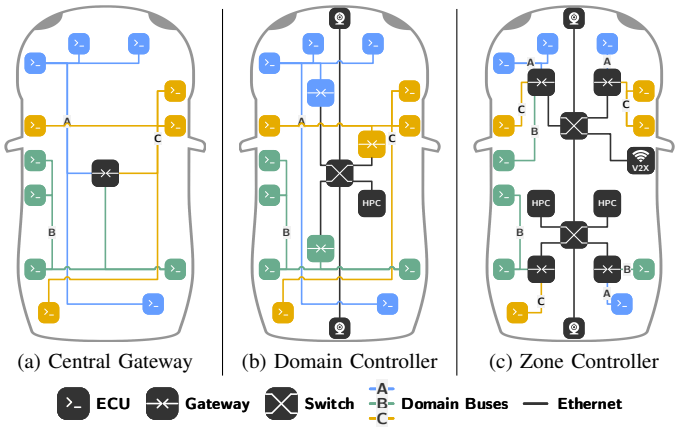


Fig. 1: Evolution of the IVN from a central gateway topology that enables cross-domain communication, to a domain controller topology that connects network domains via an Ethernet backbone, to a zone topology that splits the domain buses and connects all ECUs to a zone controller in their vicinity.

can be reduced. AUTOSAR paves the way for the transition to SOA with Scalable service-Oriented MiddlewarE over IP (SOME/IP) [15]. Containerized services are under discussion for larger ECUs [16]. This poses challenges for the network as safety-critical traffic shares a wire with other traffic.

B. Time-Sensitive Networking in Cars

In-vehicle communication requires robust QoS for simultaneous real-time and BE traffic. There are several proposals to add real-time capabilities to Ethernet, of which the standards for Time-Sensitive Networking (TSN) (IEEE 802.1Q [3]) are the most promising candidate for deployment in vehicles [17].

1) Real-time control: On the input and output ports of TSN devices, gates can be opened to let frames through, or closed to block them. A periodic Gate Control List (GCL) schedules at what time a gate opens and closes. Meters at the ingress and shapers at the egress complement the GCL functionality.

Ingress control (IEEE 802.1Qci) filters incoming frames per stream and already discards packets that missed their time slot. For frames that pass the ingress control, the switching fabric decides to which output ports they are forwarded. Egress control (IEEE 802.1Qbv) assigns frames to one of 8 priority queues, each of which has a gate and may have a selection algorithm, e.g., for bandwidth control. The first frame from the queue with the highest priority whose selection algorithm and gate allow the frame to pass is selected for transmission.

2) Communication classes: Real-time communication can be synchronous (Time Division Multiple Access (TDMA)) or asynchronous as already defined in the TSN predecessor Audio Video Bridging (AVB), which we compared in former work [18]. Asynchronous real-time communication requires bandwidth reservation of a predefined data rate per flow. The bandwidth usage is controlled by the selection algorithm of each queue at the egress, e.g., by Credit Based Shaping (CBS). Synchronous real-time communication can be implemented with a network-wide TDMA schedule for the GCLs. This can minimize latency and jitter of real-time traffic, but requires high-precision time synchronization (802.1AS-2020).

3) Schedule configuration: Scheduling in TSN has been explored in the past [19] and applied to IVNs [20]. In a scheduled network, all endpoints and switches implement TDMA. Schedules can include any selection of priorities, allowing concurrent synchronous and asynchronous communication.

The calculation of a TDMA schedule is complex and computationally intensive, therefore schedules are usually computed offline. Still, a schedule can be updated during runtime [20]. The NETCONF protocol (RFC 6241 [21]) can be used with YANG data models (RFC 6020) for reconfiguration of TSN modules (e.g., 802.1Qcp and P802.1Qcw).

There are still open challenges, such as simplifying reconfiguration, and improving security to detect threats and initiate countermeasures [17]. In this paper, we introduce dynamic traffic control for TSN by integrating it with SDN and propose security mechanisms to protect in-vehicle communication from interference with unknown traffic.

C. Software-Defined Networking in Cars

SDN [4] separates the control logic (control plane) from the underlying switches that forward the traffic (data plane) [22]. Network devices become simple forwarders that are programmed by a central SDN controller with global network knowledge using open standards such as the OpenFlow protocol [23]. Controller applications implement the behavior of the network, e.g., routing protocols. OpenFlow switches forward incoming packets based on a programmable flow table. A flow entry matches a subset of Layer 2 to Layer 4 header fields and contains actions, such as discard, forward, or modify.

1) Real-time capability: SDN is considered generally suitable for real-time environments in terms of network configuration latency [24], in particular when all flows are predefined in the switches. Still, SDN needs to be extended to control bandwidth reservation and scheduling for real-time flows.

Nayak et al. [8], [25] first mention Time-Sensitive Software-Defined Networking (TSSDN) with dynamic scheduling and routing techniques during runtime to improve robustness in TSN, but only the hosts are scheduled and not yet the switches. Thereby cross traffic can still impact the performance of real-time traffic classes, as we investigated in former work [26]. We argue that a vital requirement for TSSDN is to adapt configurations of network devices to changing real-time traffic. Earlier [6], we integrated SDN and TSN for stream reservation with OpenFlow without delay penalty for time-sensitive in-vehicle communication. Gerhard et al. [27] implemented a similar concept in a hardware environment of industrial plants. Nam et al. [28] optimized TSN stream reservation with SDN and could reduce the communication overhead for stream reservation. Corresponding work has focused on either TSN scheduling techniques (e.g., [20]) or dynamic reservation with the SRP. In this paper, we extend our approach to TSSDN by closing the gap left when updating the schedule in network devices for synchronous traffic. This work completes the design of a fully programmable TSSDN for IVNs.

Transactions are commonly used to coordinate critical changes in distributed systems. The ACID properties (atomicity, consistency, isolation, and durability) guarantee consistency of transactions despite possible errors. ACID transactions can be used in SDN environments to maintain valid state of the network [29], [30]. The impact of transactional network updates on real-time traffic has not been investigated yet. We compare two methods for transactional network updates in real-time systems, which both maintain a consistent state across distributed network devices.

2) Automotive use-cases: SDN promises to reduce complexity and increase the adaptability of networks [22]. Halba et al. [5] showed how SDN can improve the safety and robustness of IVNs through dynamic rerouting. Haeberle et al. [7] presented an IVN concept that reduces the complexity of the vehicular E/E architecture based on SDN. In previous work [31], we evaluated the performance of existing SDN controller implementations with respect to IVN requirements and found that all implementations lack important safety requirements, such as guaranteed response times, but could confirm that SDN controllers can be used in vehicles with the right modifications. To the best of our knowledge, no related work has analyzed the use of SDN to secure the IVN.

In this paper, we investigate how SDN flow control can improve the security of the IVN by precisely separating in-vehicle communication. Combining TSN and SDN can ensure that real-time requirements are met.

D. In-Vehicle Network Security

Current vehicles are vulnerable to manipulation by third parties, which has been demonstrated in the field [10]. Checkoway et al. [32] provide a fundamental analysis of the automotive attack surface and systematically show how a variety of interfaces can be used to gain malicious access to in-car devices. Manipulation of the IVN and its ECUs can compromise the safety of the vehicle, putting passengers at risk.

1) Security assessment: Assessing security mechanisms is difficult because the risk of unknown vulnerabilities is hard to predict [33]. A common method for evaluating IVN security is to analyze dark-side scenarios [34], which are based on known vulnerabilities and attack targets. These can be used to assess the probability, severity, and controllability of attacks [35]. Based on an attack tree created from real incidents, Longari et al. [36] evaluated how CAN network topologies can be hardened by introducing additional gateways. In this work, we systematically analyze the separation of in-vehicle control flows in Ether-networks and use attack scenarios to show the benefits and limitations of our network security mechanisms.

2) Taxonomy of attacks and defenses: Attacks on IVNs include Denial of Service (DoS), replay, spoofing, malware, and falsified-information attacks [14], [37]. They can be grouped in *alter* attacks that aim to modify data, *listen* attacks that aim to monitor data, *disable* attacks that aim to deny services, and *forge* attacks that aim to insert incorrect data [35].

Defensive measures can be divided into attack prevention, detection, and mitigation [37]. The main security goals for IVNs are (i) *availability*, which ensures that resources and services are accessible; (ii) *integrity*, which ensures the accuracy and completeness of data; and (iii) *authenticity*, which aims at the verifiability of data sources and sinks [14], [35].

3) Security mechanisms in IVNs: Firewalls and access control mechanisms can prevent attacks with stateful inspection,

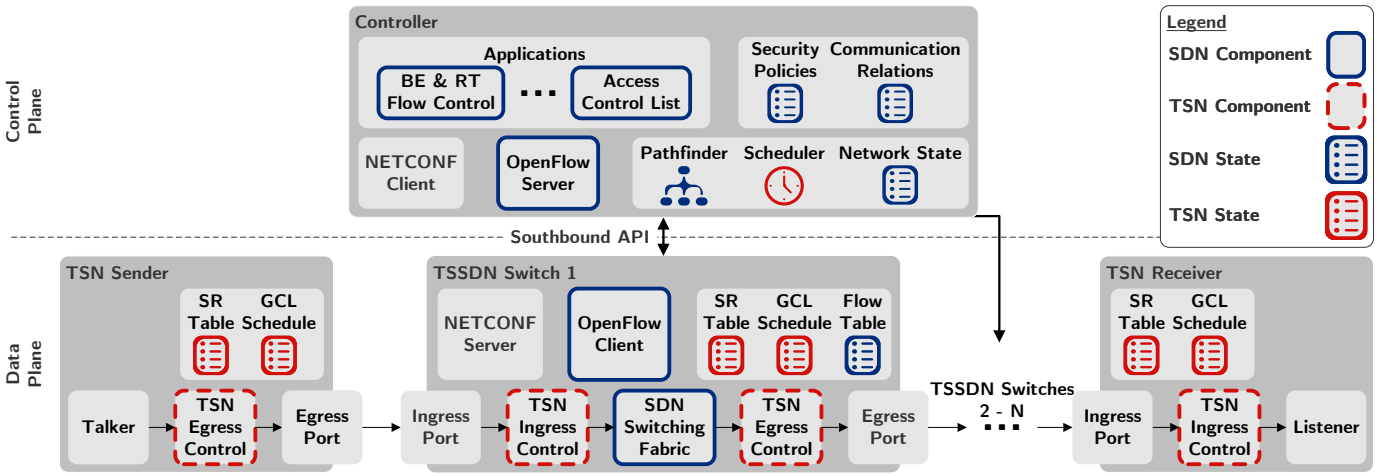


Fig. 2: Integrated network architecture of TSN and SDN. At the **data plane**, packets flow from a sending to a receiving TSN endpoint through switches that combine TSN ingress and egress control with SDN forwarding. At the **control plane**, TSN functions such as scheduling are integrated with common SDN controller functions such as path finding using a global network view. Network applications control real-time and best-effort flows simultaneously and enforce security policies.

rate limiting, and filtering [38]. Gateways can filter messages between different sections of the IVN [39] and analyze event chains based on behavior specifications [40]. In this work, we focus on network defense mechanisms that rely on precise SDN-based flow separation in the forwarding pipeline without additional processing resources.

Anomaly detection can identify ongoing attacks in the network [41], e.g., using machine learning algorithms to detect anomalies on the CAN bus [42]. In previous work [43], we presented an anomaly detection system using the TSN ingress control that can operate with zero false-positives based on a precise traffic specification. The reported security incidents from different detection mechanisms can be combined [44] and analyzed for entire vehicle fleets in the cloud to detect correlations, e.g., in terms of affected devices [45].

Integrity and authenticity can be ensured with authentication and encryption methods suitable for IVNs [9], [46], across all communication systems, such as CAN [47], FlexRay [48], and Ethernet, e.g., MACSec (IEEE 802.1AE [49]) on the MAC layer. Identity and credential management enables the verification of entities such as ECUs [50]. Encryption, however, cannot protect data flows from all network attacks. In-vehicle ECUs have limited computing power which limits the resources for security features. In addition, established and verified ECUs are used over many vehicle generations. Our network-centric approach helps to secure ECUs with limited computing power and allows secure reuse of legacy devices while remaining compatible to application layer encryption.

4) Software-Defined Networking: Flexible security solutions will be beneficial to cope with the growing dynamics of the IVN [51]. Software-defined security elements can be easily adapted to the situation at hand [14]. SDN can improve network security by detecting and preventing attacks [12]. In previous work [13], we showed how the precise flow control of SDN can be used to protect IVN control communication that is precisely specified in the communication matrix.

In this work, we use SDN mechanisms that protect IVN communication from malicious access. Enabling detection of unknown flows and countermeasures through reconfiguration.

III. Time-Sensitive Software-Defined Networking

Time-Sensitive Software-Defined Networking (TSSDN) integrates the TSN and SDN concepts on the control and data plane as shown in Figure 2. On the data plane, forwarding follows TSN ingress and egress control together with the SDN flow switching rules. On the control plane, real-time traffic control commands for switches are perceived and processed by the SDN controller. An open southbound API enables vendor-independent programming of TSN and SDN components.

A. Data Plane Architecture

The TSSDN data plane connects the TSN endpoints via switches that integrate the SDN forwarding pipeline with the TSN real-time control. Each switch contains a flow table, an Stream Reservation (SR) table, and a Gate Control List (GCL).

Frames arrive at the TSN-controlled ingress, which applies filters and time checks. After a frame passed the ingress control, the SDN switching fabric performs a lookup in its flow table for forwarding. If a matching entry is found, the predefined actions are performed and the packet is forwarded to its specified ports. The TSN egress control of the port then shapes the outbound traffic. If no matching rule exists, the packet is discarded by default. Most controllers, however, install a rule to receive these packets.

B. Control Plane Operations

The SDN paradigm mandates that control plane functions are offloaded from the switches to a central controller. The controller combines common SDN tasks such as address learning and routing with TSN functions such as stream reservation and scheduling. Network designers implement applications to control the behavior of the network through the controller using OpenFlow or NETCONF. A network application can react to messages and push updates to the data plane, thereby

leveraging the abstract network view, and knowledge about the IVN as described by the communication matrix and security policies. TSN traffic differentiation will operate as follows.

1) Best-effort flow control: BE flow control remains unaltered to traditional SDN. OpenFlow-enabled switches forward packets of unknown flows to the controller. Network applications decide whether to discard the packet, reply directly, or forward it. For the latter, the application determines a route and installs flow rules on the data plane. Thereafter, network devices can forward packets of this flow independently.

2) Asynchronous real-time flow control: For asynchronous real-time flows, senders and receivers announce their resource requirements across the network using the Stream Reservation Protocol (SRP). Originally, TSN uses a fully distributed control plane of the switches. Talkers announce streams by a *Talker Advertise*, which contains information about the stream and its bandwidth demands. Switches update their SR table and re-broadcast the announcement. Hosts willing to subscribe to a stream send a *Listener Ready* to the talker, and all devices along the path reserve bandwidth for the stream if available.

The conceptual architecture of the centralized stream reservation model (802.1Qcc) harmonizes well with the SDN paradigm. A Central Network Controller (CNC) signals the stream reservation while the communication mechanism between the controller and the network devices is not specified. We map this centralized model onto the OpenFlow protocol, detailed in [6]. Again, talkers and listeners announce streams using the SRP. Network devices forward all SRP packets to the SDN controller. A network application checks whether the available bandwidth suffices and creates a flow entry that matches the stream. Flow entries are updated as listeners leave or join. Our controller implementation uses an OpenFlow experimenter extension to reserve bandwidth in the SR table on the switches. With this, forwarding devices can identify the stream, forward it correctly, and control the bandwidth on the egress ports, e.g., with Credit Based Shaping (CBS).

3) Synchronous real-time flow control: Synchronous real-time flows are coordinated between different transmitters across multiple links. Senders periodically transmit frames of a known maximum size in their time slot. A periodic Gate Control List (GCL) schedule opens and closes specific priority gates of each output port. Time slots are shifted for devices along the path according to the packet transmission delay, which enables minimal end-to-end latency and jitter, but requires high-precision time synchronization (802.1AS-2020).

The complex calculation of TDMA schedules is commonly performed offline. Such a static schedule is not efficient for bandwidth usage, since bandwidth remains reserved even if synchronous services are not running. In addition, communication changes are not supported but may occur after updating or transitioning applications between devices.

In TSSDN, the controller can dynamically (re)calculate the GCL schedule and paths for all flows when synchronous traffic changes. Table I lists the four basic operations of changing the network configuration along with its mandatory order of execution, which we discuss in detail below. Time slots can be added or removed, and moved forward or backward within the period so that traffic is transmitted earlier or later. More complex operations can combine these operations, such as rerouting a flow (add and remove) or shifting time slots to make room for a new flow (shift and add).

TABLE I: Basic operations for configuring scheduled traffic and their execution order on network devices along the path.

Schedule update basic operation	Order of execution
Add flow and new time slot	From destination to source
Remove flow and existing time slot	From source to destination
Shift time slot to earlier point in the period	From source to destination
Shift time slot to later point in the period	From destination to source

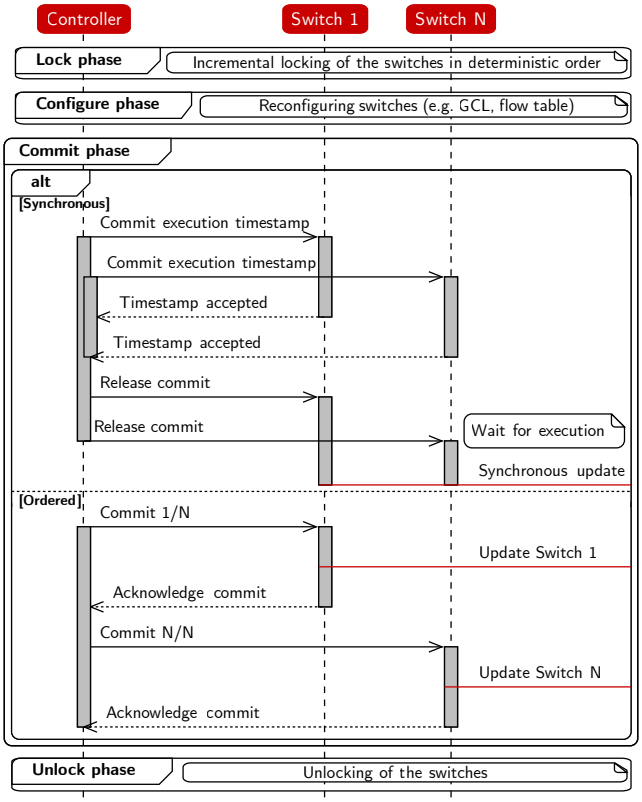
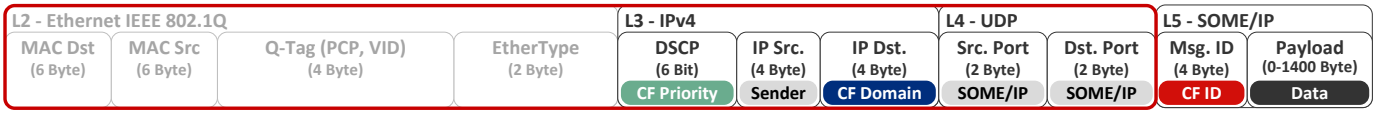


Fig. 3: Synchronous and ordered network-wide transactional update sequence.

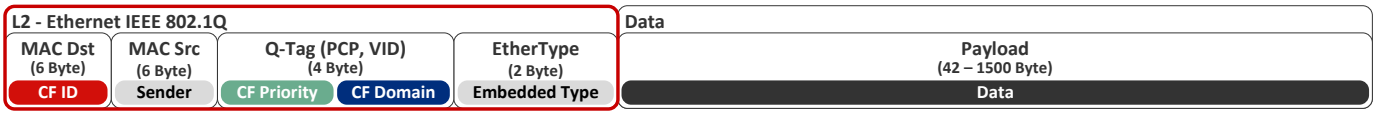
C. Transactional Updates for Real-Time Communication

The GCL is scheduled per priority queue, not per flow. Adding flows to TDMA-scheduled priorities without updating GCL can lead to overflowing queues and missed deadlines for critical traffic. Thus, the dynamic nature of SDN flow control is inappropriate to control synchronous real-time communication. In contrast, the NETCONF protocol is particularly suitable as it supports transaction-oriented configurations. Device configurations can be locked to guarantee isolation. A candidate configuration holds a copy of the device configuration and can be modified and validated before applying the changes. If an error occurs, the entire transaction is rolled back, maintaining the previously valid device state; a commit applies the changes to the running configuration otherwise.

NETCONF transactions, however, only account for operations on a single device. Transactions that span multiple devices are required to avoid queue overflow and packet loss during schedule reconfiguration. For example, if a time slot is moved to an earlier point in the schedule while a packet



(a) Hidden embedding inside a SOME/IP application layer tunnel — the network cannot match the hidden control flow identifier.



(b) Exposed embedding in the Ethernet header — the network can match the full control flow context.

Fig. 4: Different strategies for embedding Control Flow (CF) context including the identifier (ID), domain, and priority on different layers. The red outline indicates the matchable header fields in the network.

is between two network devices, the packet will miss its time slot on the next device. The packet is then sent in the next cycle, causing a delay for all upcoming traffic of that priority.

The TSSDN controller manages network-wide transactions in four phases (see Figure 3): *lock*, *configure*, *commit*, and *unlock*. All devices are locked in deterministic order (e.g., by MAC address) to prevent deadlocks. Candidate configurations are configured and validated on all devices. The commit is then orchestrated and all devices are unlocked afterwards. We identified two ways to coordinate a commit across devices consistently without packet loss or delays during configuration.

For **synchronous updates**, the controller calculates the activation time of the changes and performs a two-stage commit, first ensuring that each switch is ready and will accept the timestamp, then releasing the commit for execution on all devices at the arranged time. The timestamp can be a period number, or an absolute, TSN synchronized time. It should be chosen so that no scheduled traffic is forwarded in the network devices during the commit. This way, synchronous updates can support all combinations of basic operations in one transaction.

For **ordered updates**, the order of commit execution is determined depending on the reconfiguration, and the controller waits for confirmation of commit execution on one device to commit the next devices. Table I shows the execution order for basic operations. Time slots are added from the destination of the new flow towards the source to ensure that no packet enters the network before its time slot has been added on all devices. Removing a time slot starts at the source and ends at the destination to ensure that no packets remain on the network. To shift a slot ahead in time, the changes proceed from the source to the destination to ensure that packets are early for their time slots rather than too late. Shifting to a later point proceeds in reverse order. This tells us that the update order is application-specific. Updates that combine multiple operations with different commit orders must be split into multiple transactions, which we analyze in Section V.

Ordered updates rely on standard NETCONF commit management, but synchronous updates also require timing the commit execution. In addition, we assume that commit execution on forwarding devices is executed atomically and always succeeds. Although both approaches guarantee interference-free communication, there are still open problems that remain for future work: The start time of a transaction must be set so that all changes proceed before the traffic starts – we set the time accordingly in our simulation. A protocol is missing for senders of synchronous traffic to inform the controller about characteristics of their traffic, e.g., packet size, frequency, start and stop times – we assume that the controller knows them.

IV. Securing Time-Sensitive Software-Defined In-Vehicle Networks

For a specific car, critical in-vehicle communication is deterministically defined, which enables the controller to steer traffic precisely based on its IVN knowledge including ECUs, control flows, and timing information from the communication matrix. SDN switches identify flows in the OpenFlow pipeline by matching packet header fields from Layer 2 to Layer 4.

A. Embedding Strategies for Control Flow Information

A **Control Flow (CF)** is a sequence of related messages with the same unique identifier in the vehicle – called a CF ID. This could be a CAN message, for example. A CF is sent from a single origin and follows a point-to-multipoint relationship to reach one or more receivers. The priority of a CF is determined by its deadline and criticality, e.g., safety critical messages have the highest priority.

Gateways forward data between bus systems and the Ethernet backbone. Therefore, the CF is repackaged, e.g., from a CAN frame to an Ethernet frame. Future Ethernet communication in vehicles can exploit the entire stack to embed control flows. Figure 4 presents two examples of CF embedding on different layers. Each CF has an identifier (ID), a sender, a priority and a domain. Depending on the repackaging, this information can be **hidden** from or **exposed** to the network.

The hidden embedding tunnels a CF using an application layer protocol, which is the current state of the art. Figure 4a shows an example that uses SOME/IP. The CF ID is encoded in the message ID field and the data is embedded as payload. The example uses reserved UDP ports for SOME/IP traffic. The Differentiated Services Code Point (DSCP) is populated with the CF priority. The Ethernet header is generated by the network stack, which can be configured to map the DSCP to a Priority Code Point (PCP) for QoS on Layer 2. All CFs in a domain are sent to the same IPv4 multicast address.

In contrast, embedding can happen in a completely exposed way. Here, the sender embeds the context of the CF only in packet header fields that are used for the forwarding decision in the network (Layer 2 to Layer 4). Figure 4b shows an example for exposed embedding in the Ethernet header. For a discussion on the advantages of an embedding on the lowest

possible layer see Section IV-D. The CF ID is encoded as a multicast destination MAC address. Virtual LANs are created to separate bus domains. The CF priority is mapped on the PCP for Layer 2 QoS differentiation. We embed the message data and length into the Layer 2 payload. The EtherType specifies the embedded data type, e.g., a custom type for embedded CAN data.

B. Separating In-Vehicle Control Flows in the Network

The embedded context information within the packet header fields as used for the forwarding decision is outlined in red in Figure 4. A **Network Flow (NF)** is a sequence of contiguous packets of one or more CFs that have identical matchable header fields and are therefore treated equally by the network. A NF is transported from a specific source to a destination in the network. Using multicast, a NF can reach multiple sinks, which is a common use case in IVNs, since the same information is often needed at different ECUs. The choice of the embedding approach is a key factor in distinguishing CFs.

With hidden embeddings, a **separation by domain** can be achieved if a domain identifier is encoded in the destination IP address. This creates a domain tunnel that forms a point-to-multipoint NF per sender and domain, which is the current state of the art. The tunnel is identified by the source and destination IP addresses and UDP ports. The NF destinations are the group of receivers of all CFs in the tunnel. The CFs of a domain cannot be differentiated by the network.

To improve the separation of CFs with hidden embeddings, a finer-grained **separation by topic** can be created. A topic can group a small number of related CFs, e.g., all communication for the engine or for lighting control. In CAN bus architectures, each CF belongs to exactly one domain bus. Introducing a new domain for CFs also means creating a new physical bus. On an Ethernet backbone, a new topic requires only a new tunnel on an existing physical link. As cross-domain communication increases, it seems advisable to form smaller groups of CFs than the original vehicle domains. The topic identifier can be encoded in the multicast IP address instead of the domain identifier so that each topic has a point-to-multipoint NF for each sender in the topic.

An exact **separation by message** type can only be achieved via exposed embedding. Each CF has its own point-to-multipoint NF, which can be identified and separated in the network by matching the multicast destination MAC address.

C. Reliability and Security Considerations

Reliable communication is essential for safety-critical traffic in the vehicle. Basic driving functionality must be guaranteed in order to achieve a safe state in an emergency, e.g., stopping at the edge of the road. A static configuration for safety-critical communication can be verified offline to ensure its correctness under all circumstances, and redundant paths can be confirmed to increase resilience.

The static flow and timing configuration is loaded in each forwarding device during boot, which reduces startup times because these flows are not set up via the controller. For changing this static configuration a firmware update is required as the SDN controller can not alter it in any way. To achieve this, a protected separate flow table can be used that is always matched before dynamic rules. For the TSN configuration, the priority queues can be partitioned in a set of static and dynamic queues [52]. This allows the addition of new flows in dynamic priorities and protects the static priorities by design. In the event of a controller failure or critical security incident, the static configuration serves as a fail-operational configuration.

Dynamic traffic that is not always needed or where the communication partners are not known before runtime can be controlled by the SDN controller. The controller can verify that the new communication is allowed and identify senders and receivers to create a precise flow that matches the header information from Layer 2 to Layer 4. In addition, bandwidth or time slots can be reserved for these flows. An Access Control List (ACL) can define additional patterns that whitelist or blacklist dynamic flows. For example, complete protocols, such as ARP or ICMP, can be (dis-)allowed in the IVN or dynamic communication can be blocked for some hosts.

D. Impact on In-Vehicle Network Design

When Control Flow (CF) information is embedded at the application layer, it cannot be used for forwarding decisions without violating the OSI layers. In SDN, exposed embedding of the CF ID in any of the matched header fields from Layer 2 to 4 will enable a precise separation by message. Still, there are several advantages of embeddings on the lowest possible layer. Layer 2 information is only valid in the local IVN, so Ethernet embeddings are not routable which can make attacks from outside the vehicle more difficult. In addition, embeddings in the Ethernet header support the use of non-SDN switches with the same separation. Embedding options are also affected by encryption, as the layers used for the forwarding decision are useless if encrypted.

When small control messages are embedded into Ethernet, their aggregation is an approach to reduce overhead. For example, a CAN message has a payload of 8 bytes, while an Ethernet frame has a minimum size of 64 bytes with a minimum payload of 42 bytes. Multiple CAN messages can be sent in the same frame to save bandwidth, but this also delays messages, increasing latency and jitter [53]. With different embedding approaches, aggregation can be hindered, e.g., exposed embeddings make aggregation impossible.

Control information is often transmitted in cycles with a similar data size. TSN schedules can be more efficient the better CFs can be distinguished in the network. With exposed embeddings, timing can be determined more accurately, resulting in lower TDMA reservation overhead. On the other hand, schedule computation overhead increases as the number of small time slots increases. Aggregation complicates timing computation due to varying sizes and intervals of packets.

V. Validating the TSSDN Switching Architecture

The key performance characteristic of the proposed TSSDN architecture is its seamless integration of SDN control with real-time communication. We evaluate this in a simulation environment (see Figure 5) based on the OMNeT++ simulator [54]. Our model Software-Defined Networking for

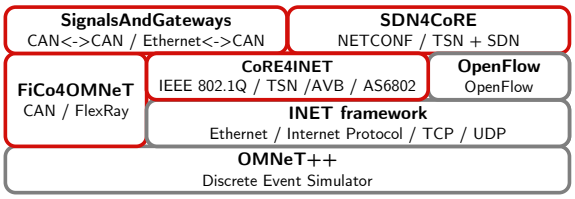


Fig. 5: Simulation environment used for the evaluation available as open source, see *github.com/CoRE-RG/SDN4CoRE*.

Communication over Real-Time Ethernet (SDN4CoRE) [55] implements the proposed architecture based on the INET framework [56], the OpenFlowOMNeTSuite [57], and our frameworks for IVNs [58]. The models outlined in red are maintained by our research group and published as open source at *sim.core-rg.de* and *github.com/CoRE-RG*. We use a specific scenario with carefully designed TDMA traffic and varying amounts of cross traffic to analyze the impact of SDN control and schedule reconfiguration on real-time traffic. This scenario is also available in the SDN4CoRE repository.

A. Network Topology and Traffic Configuration

Figure 6 shows our network for the evaluation. It consists of two TSSDN switches, one SDN controller, and four sources (S1-S4) sending to one destination. All links have a bandwidth of 100 Mbit/s. With this setup we can analyze the timing of real-time communication under the load of cross traffic across multiple links and verify that timing requirements of TSN are also met in TSSDN. For comparison, we consider the identical scenario with pure TSN switches that perform stream reservation in a distributed manner and have a pre-configured TDMA schedule, which is the current state of the art.

All sources send BE traffic (PCP 0) with varying cycle and frame sizes. S1, S2, and S3 send synchronous traffic each at an individual priority (PCP 5 to 7), which are scheduled in a TDMA fashion with a period of 1 ms. S1 and S2 send one maximum Ethernet frame per period and S3 sends two. S4 sends a maximum Ethernet frame every 1 ms as asynchronous traffic of medium priority (PCP 4) directly after the Stream Reservation (SR) is completed.

The timeline in Figure 7 visualizes the start and stop times of real-time communication. For asynchronous traffic, the controller performs the SR using OpenFlow. Prior to changes in synchronous traffic, we assume that the SDN controller has been informed and the Gate Control List (GCL) schedule is reconfigured (C1 – C6) via NETCONF in the period before the first or after the last frame of the altering flow. In the pure TSN version, the GCL schedule is statically configured and the switches perform the SRP of TSN independently.

Our scenario contains all basic operations for reconfiguring the network, which follows from the changes in synchronous real-time communication. Synchronous traffic is added to an empty schedule (C1) and to an existing schedule (C2). The time slot for S3 is inserted between the time slots for S1 and S2 (C3), so that the time slot for S2 needs shifting to an earlier time and for S1 to a later time in the period. This allows us to analyze the impact of combining multiple basic operations with different mandatory commit sequences in one transaction. Each synchronous transmitter stops after sending for 300 ms.

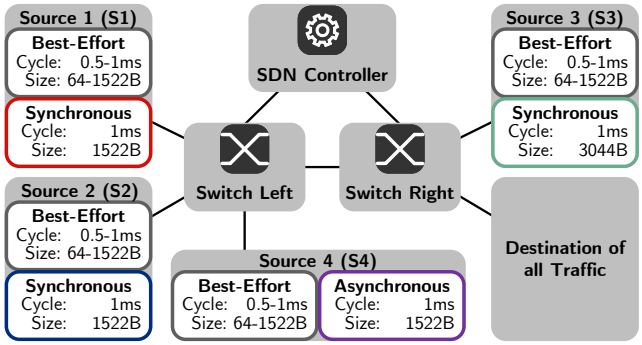


Fig. 6: Evaluation network with two TSSDN switches, one SDN controller, four sources and one destination connected via 100 Mbit/s links. Sources 1 to 4 send best-effort, synchronous and asynchronous real-time traffic that differs in transmission cycle and Ethernet frame size.

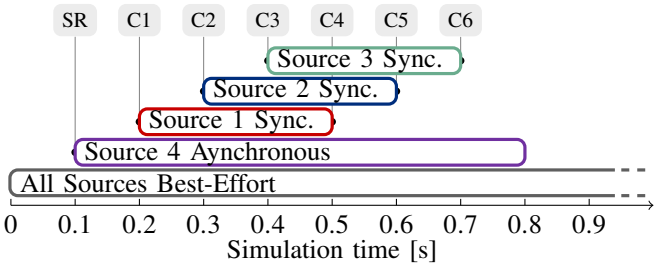


Fig. 7: Timeline indicating the start and stop times of traffic with their stream reservation (SR) or configurations (C1–C6).

The network removes time slots from existing schedules (C4, C5) and returns to an empty schedule at the end (C6).

B. TDMA Schedule and Worst-Case Analysis

We determine periodic TDMA schedules that coordinate synchronous flows in the 1 ms period. The transmission windows of synchronous traffic at each source are shown in Figure 8 for the entire period. Figure 9 displays the GCL schedule of the two switches. The pure TSN network is preconfigured on startup with configuration C3, which includes all synchronous flows. In the TSSDN variant of the network, the GCL configuration is updated according to the active synchronous traffic (C1 to C6). At the beginning, all gates are open because there is no synchronous traffic yet.

For each sender, our GCL configuration exclusively assigns one time slot to its 802.1Q priority. A Guard Band (GB), in which all gates are closed, is added to ensure that synchronous traffic cannot be delayed by other traffic. All gates are closed so that the transmission time of a maximum Ethernet frame (1522 B + 8 B preamble) can be completed. This takes 123.36 µs on a 100 Mbit/s link, including the inter frame gap (T_{ifg} = 0.96 µs). To ensure timely transmission, the GCLs at the senders are also scheduled with a GB before their transmission windows (see Figure 8).

Synchronous traffic, which is precisely scheduled, cannot be delayed as it is protected by a GB. Its maximal end-to-end latency T_{L} can be analytically calculated equivalent to the best-case latency when all clocks are perfectly synchronized.

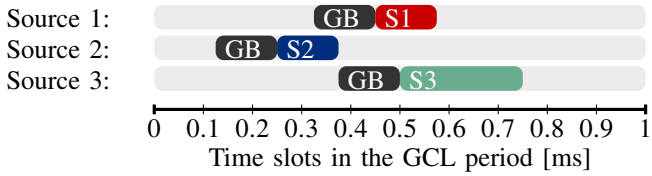


Fig. 8: Transmission windows of the synchronous sources in the period of the Gate Control List (GCL) schedule.

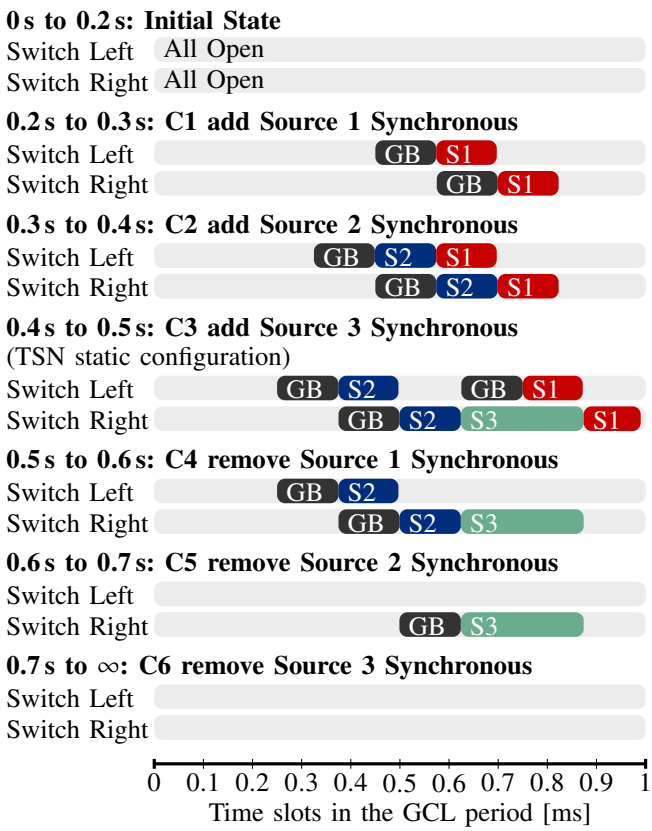


Fig. 9: Gate Control List (GCL) schedule of the two switches, which is reconfigured according to the changes in synchronous traffic (C1 to C6). The time slots for S1, S2 and S3 are scheduled along their path. Guard Bands (GB) with all gates closed are added to prevent delays from competing traffic. The TSN-only version is statically configured as depicted in C3.

In our evaluation, we take a closer look at the latency of S1. The calculations for S2 and S3 are not shown in detail.

Reconfiguration C1 adds the synchronous flow from S1 at 0.2 s simulation time. S1 transmits one full size frame 450 µs after the start of each period. A time slot is reserved along the path. After the delay for transmission (T_{trans} = 122.4 µs) and forwarding (T_{fwd} = 3 µs) the frame arrives at the output of Switch Left at 575 µs and its time slot begins. A GB is inserted before it. All gates are opened again when the frame completes transmission at 700 µs and its time slot at Switch Right begins. It ends at 825 µs and the GCL is configured accordingly. The analytic end-to-end latency bound for synchronous traffic from S1 with configuration C1 ($T^{\text{S1}}_{\text{L-C1}}$) reads:

$$\begin{aligned} T^{\text{S1}}_{\text{L-C1}} &= 3 * T_{\text{trans}} + 2 * T_{\text{fwd}} \\ &= 3 * 122.4\,\mu\text{s} + 2 * 3\,\mu\text{s} = 373.2\,\mu\text{s} \end{aligned} \tag{1}$$

The controller activates C2 when the synchronous traffic from S2 starts at 0.3 s simulation time. S2 sends a full size frame 250 µs after the start of each period. Thus, traffic from S2 arrives at 375 µs at the output of Switch Left. To save bandwidth the time slot of S2 is scheduled to finish exactly before the time slot of S1. The GB is shifted accordingly. This introduces a queueing delay of 75 µs. The frame arrives at Switch Right at the beginning of its time slot to be transmitted. This results in a latency bound of $T^{\text{S2}}_{\text{L-C2}}$ = 448.2 µs.

The schedule is reconfigured for S3 at 0.4 s simulation time (C3). S3 sends two full size frames 500 µs after the start of each period. This new schedule minimizes the latency of S3 ($T^{\text{S3}}_{\text{L-C3}}$ = 247.8 µs) by adding a time slot on Switch Right from 625 µs until 875 µs. This requires to shift the time slots of S1 to a later time and S2 to an earlier time, which also changes their latency bound. The time slot for S2 is now 75 µs earlier on both switches which also makes up for the queueing delay and reduces the latency bound to $T^{\text{S2}}_{\text{L-C3}}$ = 373.2 µs. The time slot for S1 is delayed by 175 µs which introduces a queuing delay (T_{queue}) and changes the latency bound to

$$\begin{aligned} T^{\text{S1}}_{\text{L-C3}} &= 3 * T_{\text{trans}} + 2 * T_{\text{fwd}} + T_{\text{queue}} \\ &= 3 * 122.4\,\mu\text{s} + 2 * 3\,\mu\text{s} + 175\,\mu\text{s} = 548.2\,\mu\text{s} \end{aligned} \tag{2}$$

Asynchronous traffic can be delayed by cross traffic and the TSN gates. We examined the coexistence of scheduled and bandwidth reserved traffic and the impact of cross traffic in previous work [26], [59]. In the worst-case, the asynchronous traffic from S4 is delayed by one full size BE frame ($T_{\text{BE max}}$) and the inter frame gap (T_{ifg}) on every hop. In our scenario, it can also be delayed through the schedule at each port by the duration of the GB and the time slots of synchronous traffic. The schedules for Switch Left and Right are aligned so that a frame cannot be delayed on both hops. For the configuration C3 this results in a maximum interference through the schedule of T_{mi} = 616.8 µs. The analytical end-to-end latency bound for the asynchronous traffic from S4 ($T^{\text{S4}}_{\text{wc}}$) reads:

$$\begin{aligned} T^{\text{S4}}_{\text{L}_{\text{wc}}} &= T_{\text{mi}} + 3 * (T_{\text{BE max}} + T_{\text{ifg}} + T_{\text{trans}}) + 2 * T_{\text{fwd}} \\ &= 616.8\,\mu\text{s} + 3 * (122.4\,\mu\text{s} + 0.96\,\mu\text{s} + 122.4\,\mu\text{s}) + 2 * 3\,\mu\text{s} \\ &= 1.36\,\text{ms} \end{aligned} \tag{3}$$

C. Impact of SDN on Real-Time Flows

Figure 10 shows the minimum, maximum and average end-to-end latency for one exemplary flow of each traffic class from 1 s simulation runs. To ensure that the data is not affected by the timing of traffic in the period, the minimum, maximum, and average from simulations with 20 different seeds for the start time of the asynchronous traffic are shown. The queues in our network are infinite so all packets will arrive at there destination eventually and no packets are lost.

All traffic shows identical or lower end-to-end latency in the TSSDN variant than in pure TSN. This is due to the re-scheduled time slots (C1 to C6) that match the synchronous traffic exactly without surplus bandwidth. Given the same configuration for both variants (after C3), the synchronous traffic of S1 has the same constant latency corresponding to the analytical bound (see Equation 2). Before C3 is applied, S1 has a smaller latency in TSSDN as calculated in Equation 1.

S4 starts sending asynchronous traffic after the Stream Reservation (SR) at 0.1 s. Again, latency is lower with the TSSDN variant because less bandwidth is reserved for the scheduled traffic. The latency varies largely for the 20 different seeds for the start time in the period, which is to be expected since the frame delays fluctuate by the schedule. Still, the maximum latency never exceeds the analytic bound of Equation 4.

The BE traffic flows without worst-case guaranties. Frame sizes vary between minimal and maximal Ethernet frames, which explains the small minimum latency. In TSSDN, BE flows experiences a larger delay if set up by the controller.

The latency for flow installation is avoided for asynchronous traffic with the Stream Reservation Protocol (SRP). As such flows are already installed in the TSSDN switches during the SR, no further inspection by the SDN controller is needed. Thus, additional latency ceases for TSSDN after the SR.

Table II compares the SR duration of TSN and TSSDN, which vary largely with the 20 seeds (uniformly distributed over the 1 ms period of the schedule). The TSSDN variant has a lower maximum delay when set up at simulation time 0.1 s, since no gate control schedule is active at this time. We also ran the simulation in TSSDN starting the SR at 0.45 s with configuration C3 active and thus the same schedule as in the TSN-only variant. The results show an additional delay in TSSDN caused by the communication between the switches and the controller. This is in agreement with our previous findings for SR in TSSDN [6] but extends insights to the impact of scheduled traffic on SR sent as BE traffic in both cases, TSSDN and plain TSN.

D. Impact of Transactional Updates

The impact of the synchronous and ordered update methods on the latency of the scheduled traffic is shown in Figure 11. We vary the commit execution time to find the worst case for each configuration method. For simple unordered reconfiguration, the result is completely random and not shown. For synchronous updates in both switches, the latency between two configurations is constant for each interval and matches the calculated bounds obtained in Section V-B. After C3, the latency of S1 and S2 changes as time slots are shifted for the additional flow from S3.

The ordered updates behave identical for all transactions except for C3, since this requires only one commit order. The update sequence for C3 is set to change Switch Left first and then Switch Right, which agrees with the commit order for shifting the time slot from S2 to an earlier time and adding a time slot for S3, but conflicts with the commit order for shifting S1 to a later time. After Switch Left is updated, a frame from S1 must wait additional 175 µs according to the new schedule. After this frame is transmitted, it misses its time slot at Switch Right, which has not yet been updated. This delays the frame by 1 ms until the next cycle, which can be observed on a few frames of S1 at 0.4 s. Then Switch Right is updated, which enforces the new schedule but cannot repair the one-cycle delay. This delays the frame by 1 ms until the next cycle, which can be observed on a few frames of S1 at 0.4 s. Then Switch Right is updated, which enforces the new schedule but cannot repair the one-cycle delay.

TABLE II: Stream Reservation (SR) delay of TSN and TSSDN w/ and w/o Gate Control (GC) for 20 different SR start times.

Variant		SR start (20 seeds; 0.05 ms steps)	SR duration Minimum	Average	Maximum
TSN	w/ GC	0.1 s to 0.100 95 s	0.115 ms	0.775 ms	1.404 ms
TSSDN	w/o GC	0.1 s to 0.100 95 s	0.828 ms	0.862 ms	1.001 ms
TSSDN	w/ GC	0.45 s to 0.450 95 s	0.828 ms	1.495 ms	2.370 ms

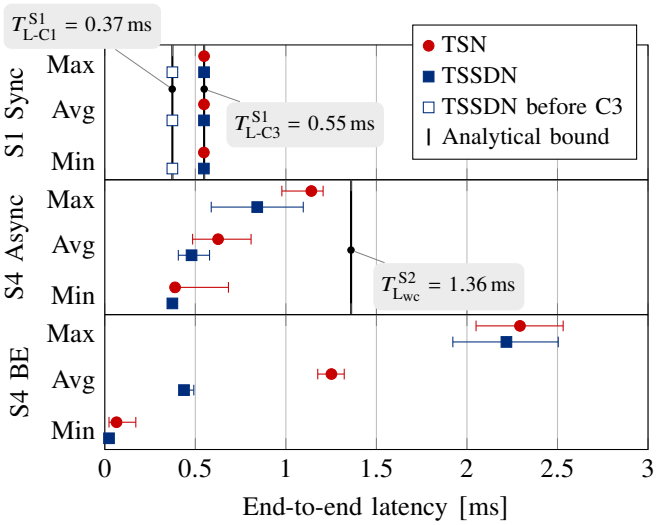


Fig. 10: End-to-end latency comparison for the three traffic classes in TSN and TSSDN. The minimum, average, maximum, and the analytical bound are depicted. For each min/avg/max, again the average and deviation for minimum and maximum is indicated from simulations with 20 seeds.

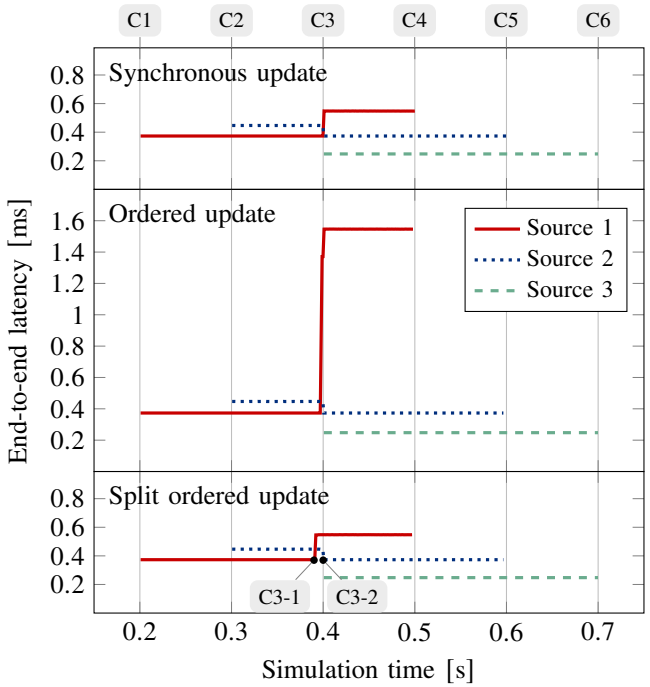


Fig. 11: Impact of reconfigurations C1–C6 on end-to-end latency of synchronous traffic for three different update methods. Synchronous update – both switches apply the changes at a synchronized time; ordered update – changes are executed in the order required by the first operation; split ordered update – configuration C3 is split in two transactions (C3-1, C3-2).

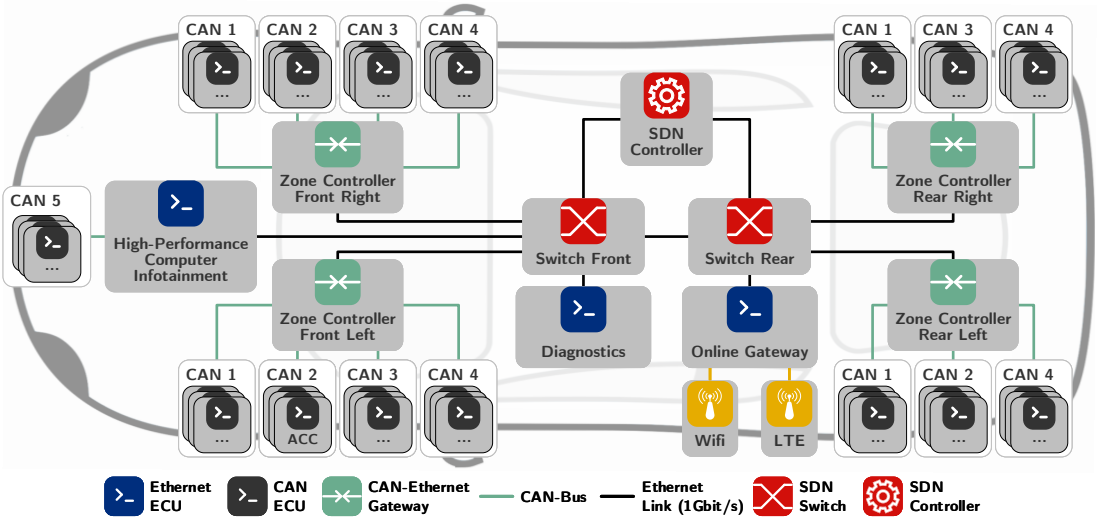


Fig. 12: IVN of a production vehicle transformed to a software-defined Ethernet network in a zone topology.

Splitting transaction C3 into two transactions (C3-1, C3-2) can solve this problem. C3-1 only shifts the flow of S1 first on Switch Right, then on Switch Left. C3-2 shifts the flow of S2 and adds the flow S4 in reverse order. This ensures the correct execution order for all operations in the transaction.

E. Discussion

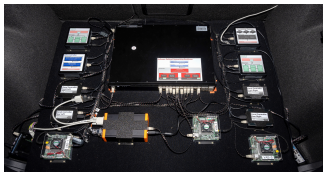
Synchronous traffic schedules allow for reconfiguration without a penalty for existing or additional real-time traffic, provided a suitable update method executes commits simultaneously on all devices or iteratively in the correct order. For the latter, the order depends on the type of reconfiguration and requires that a transaction contains only operations of the same update priority. Synchronous updates can bundle all operations within one transaction.

Controlling asynchronous communication via OpenFlow does not affect real-time traffic, only the SR experiences a small start up delay. Since the SRP is transmitted as BE traffic, it may be affected by cross traffic, and hence stays without guaranteed temporal bounds in both the TSSDN and TSN-only variants. Our additional delay of 0.97 ms is acceptable as it stays well below the automotive requirements for service setup times, which range around 150 ms to 200 ms. To leverage the full potential of SDN, the SRP can be optimized so that the controller does not need to propagate between switches [28]. This could result in an even lower startup delay.

A side effect observed in our evaluations is that SDN protects real-time traffic. Precise flow matching acts as an ingress control and ensures that new traffic is not forwarded until a flow rule has been installed. For transactional schedule reconfiguration, this also implies that the GCL has already been updated prior to accommodating a flow. Unknown flows are not queued and thus cannot delay existing real-time traffic.

Finally, our results show that the combination of SDN and TSN in the proposed architecture works as expected. Deadlines for synchronous and asynchronous TSN flows are met and remain unaffected by the introduction of SDN. Meanwhile, forwarding is controlled by an SDN controller, which opens potentials for resilience, security, and adaptability of the IVN.

(a) Seat Ateca prototype vehicle



(b) Installation in the trunk

Fig. 13: Pictures of the prototype and installed components.

VI. Security Impact of Precise SDN Flow Control for In-Vehicle Communication

The goal of our security approach with TSSDN is to isolate in-vehicle control flows in a shared environment and prevent unwanted traffic. We compare the different flow separation concepts described in Section IV-B in a case study of a realistic IVN that was derived from a production car.

The attacker model for our case-based security study focuses on remote attackers. A remote attacker needs access to the car first to launch an attack. There are many interfaces, most of which are connected to larger ECUs such as the infotainment or an online gateway [32]. Once the attacker manages to overcome the first layer of defense, he can gain access to the IVN backbone to launch attacks on in-vehicle components. Correspondingly, our threat model reflects network-centric attacks. Threats arrive from scanning, denial of service, replay, and forgery attacks. We elaborate in characteristic examples, against which threats the TSSDN flow separation can safeguard and against which it cannot.

A. Experimental Setup of Our SDN-Based Prototype Car

Figure 13 shows our prototype car (13a) with a software-defined Ethernet backbone installed in the trunk (13b). The network topology is shown in Figure 12 derived from the CAN network originally implemented in a central gateway topology, which we transformed into an Ethernet zone topology. All CAN ECUs are grouped into four zones based on their placement in the vehicle (Front-Left (FL), -Right (FR), Rear-

Left (RL), -Right (RR)). A Zone Controller (ZC) handles computation intensive tasks and acts as a CAN-Ethernet gateway.

Despite the placement of the CAN-ECUs, the original domain buses (numbered from 1 to 5) are retained, e.g., CAN 1 on ZC FL and CAN 1 on ZC FR originally belonged to a single domain bus. A High-Performance Computer handles the infotainment system and acts as a gateway between all CAN ECUs and the infotainment domain. We extend the original CAN network of the production car with Ethernet ECUs for diagnostics and external connectivity.

The software-defined Ethernet backbone consisting of two switches controlled by an SDN controller. We generate three network configurations that correspond to our three separation concepts compliant to the communication matrix of our original production vehicle. In one configuration, ZCs embed CAN messages fully exposed on Layer 2 to enable separation by messages. The other two configurations use hidden embedding in SOME/IP tunnels and encode the topic or domain identifier in the multicast destination IP, enabling separation by topic or domain. The receiving ZC transforms the packets back to CAN frames and forwards them to the CAN bus destinations.

In all three configurations, the original messages from the vehicle are correctly forwarded to all valid receivers. Our analysis focuses on the invalid control flows that are either blocked or forwarded by the backbone. The flow separation is not affected by the amount of data transported in the flows and does not depend on the state of the vehicle, e.g., whether it is driving or not. In this way, we assume the worst case scenario which allows all original vehicular control flows at all times. Configuring the network according to the vehicle state can reduce the number of legible flows in the network, which enhances the effect of flow separation, but will not change the characteristic methods of separation.

B. Mapping Control Flows on Network Flows

We focus our analysis on the CAN Control Flows (CFs) that traverse the Ethernet backbone. Therefore, we do not consider local CFs, for which the sender and all receivers are located within the same zone. In total there are 242 different CFs forwarded via the backbone.

TABLE III: Control Flows (CF) bundled in a Network Flow (NF) with the number of NFs that carry multiple CFs.

Separation	**# NFs (with multiple CFs)**	**# CFs per NF**		
		Minimum	Average	Maximum
By Message	242 (0)	1	1	1
By Domain	19 (19)	5	13	37
By Topic	102 (38)	1	3	17

Table III shows the generated Network Flows (NFs) for each separation concept in relation to the number of CFs bundled in a NF. Message separation isolates the 242 CFs within individual NFs. Separation by domain generates 19 NFs, one tunnel for each sender and domain all of which carry multiple CFs. There is a minimum of 5, an average of 13, and a maximum of 37 CFs per NF, so there is at least one NF carrying 37 CFs that cannot be distinguished by the network.

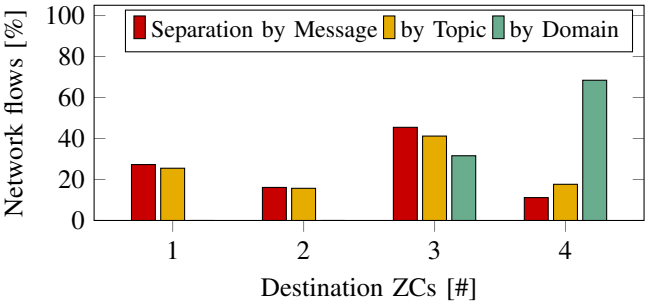


Fig. 14: Destination ZCs reached by a share of NFs.

The topic separation generates 102 NFs, one tunnel for each sender and topic. 38 of these NFs carry multiple CFs. This means that 64 of the topics contain only one CF and therefore behave exactly as if separated by message. In the following analysis, separation by message can be used as a baseline as it implements exactly the relations of the communication matrix.

All NFs are implemented as multicast to reach multiple destinations if necessary. Figure 14 shows the share of NFs sent to a particular number of destinations (ZCs). Separation by message and topic show similar results, although additional 7% of NFs reach all gateways with separation by topic. Again, this similarity could be caused by the 64 CFs in completely isolated topics. With domain separation zero NFs reach fewer than three gateways, around a third reach exactly three and nearly 70% reach all gateways in the network. In particular, the relative difference of NFs reaching all four destinations is notable. This is expected as the ECUs of a domain are usually distributed throughout the vehicle and therefore most domains are present in every zone. Nevertheless, this is a security issue as those receivers should not be able to receive those flows and also indicates that the embedding has a big influence on network overhead.

C. Security and Performance of Control Flow Separation

The attack surface of CFs depends on their isolation in the network. We compare the separation concepts based on how well CFs can be attached to paths between the ZCs. Without separation, any node in the network could send and receive any of the 242 different CFs traversing the backbone. Optimally, however, only gateways that actually link ECUs participating in a CF should be able to send and receive it on the backbone. To evaluate this, we sort the 242 possible CFs that could be sent from one ZC to another into the following categories:

- A CF is **legitimate** if the original CAN source ECU is connected to the source ZC and at least one CAN ECU connected to the destination ZC is a valid receiver according to the communication matrix.
- An **oversupplied** CF is legitimately sent but received by a destination ZC that does not need it.
- A CF is **permitted** if it could be sent by the source and would be forwarded to the destination. Even though not present in our communication matrix, these flows could be used by malicious components.
- A **forbidden** CF is filtered by the backbone and is not forwarded from the source to the destination.

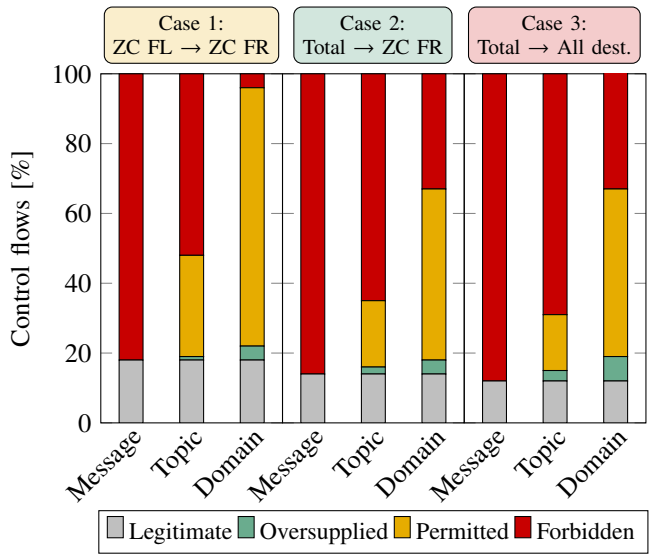


Fig. 15: Share of legitimate, oversupplied, permitted, and forbidden CFs from ZC FL to ZC FR, all sources to ZC FR, and all sources to all destinations.

Figure 15 visualizes the shares of legitimate, oversupplied, permitted and forbidden CFs in relation to the maximum possible CFs in three cases: (1) concentrates on a specific connection between two exemplary ZCs (from ZC FL to ZC FR), (2) includes all CFs from any source ZC to one destination (ZC FR), and (3) looks at the whole backbone communication from all ZCs to all ZCs. An exhaustive analysis of all communication relations between the gateways in our prototype can be found in [13].

All three cases clearly show the effectiveness of the strict message separation from exposed embedding. When separated by message, the software-defined backbone only forwards the legitimate CFs and forbids any other CF. The hidden application layer embeddings oversupply and permit illegitimate CFs. Still, only CFs of the tunnels that are already in use by the sending nodes are supported by the network. More precisely, topics can improve the separation of flows in the network. As explained earlier, we chose our topics by grouping similar messages from the communication matrix. We could not group some CFs into topics, making them perfectly separated in their own topic, which affects the results.

In the third case, 12% of all CFs from all sources to all destinations are legitimate. With separation by topic additional 3% are oversupplied at the destinations, with domain separation around 7%. Gateways can filter the unneeded CFs at the cost of processing power so that no unwanted message reaches a CAN bus. Still, the network load on the backbone increases through oversupplied control traffic with domain and topic separation. This clearly conflicts with the benefits of aggregation, which is one of the main arguments for using hidden embeddings.

Table IV shows the average bandwidth sent and received by all four ZCs and the frame size of Ethernet-embedded CAN traffic from our production vehicle over a period of 60 s. With exposed and hidden embedding, about the same bandwidth is sent without aggregation. The average frame size increases when multiple CAN frames are aggregated for a given interval

TABLE IV: Sent and received bandwidth at all four ZCs of Ethernet embedded CAN traffic (CAN Bus 1 – 4) separated by message or domain with different aggregation intervals.

Embedding	Aggregation interval	Average frame size	Bandwidth sent	Bandwidth received
Per message	w/o agg.	64 B	1.4 Mbit/s	3.1 Mbit/s
Per domain	w/o agg.	72 B	1.5 Mbit/s	4.6 Mbit/s
	1 ms	101 B	1.1 Mbit/s	3.3 Mbit/s
	5 ms	119 B	1.0 Mbit/s	2.9 Mbit/s
	10 ms	145 B	0.9 Mbit/s	2.6 Mbit/s
	50 ms	324 B	0.7 Mbit/s	2.1 Mbit/s
	100 ms	527 B	0.6 Mbit/s	1.9 Mbit/s

before being sent alone or together with other messages, which in turn significantly increases latency and jitter. This is not possible for exposed embeddings, since only one message per frame is allowed. Most evaluations of aggregation strategies only consider the bandwidth sent by the gateways. Our evaluation shows that the embedding strategy and thus the enabled separation in the network has a major impact on the bandwidth at the receiver side. Without aggregation, separation by domain results in 50% higher received bandwidth, again illustrating oversupply. The transmitted and received bandwidth depends on the traffic pattern and network topology. Kern et al. [53] were able to reduce the transmitted bandwidth by up to 75% using synthetic CAN traffic when messages are aggregated for 5 ms to 10 ms. For our specific vehicle, aggregation for an interval of 10 ms reduces the bandwidth sent by 35% while the received bandwidth is reduced by only 6%. The evaluation shows that aggregating traffic from low-bandwidth bus systems such as CAN has little impact, especially when considering 1 Gbit/s Ethernet networks. Above all, it is a security issue that network nodes are confronted with messages for which they are not prepared. Separation by message is the only concept that can solve this problem with zero oversupplies.

From a security perspective, the CFs that a (malicious) ECU could send are even more relevant than the CFs it can receive. From all sources to all destinations, 31% of all CFs are permitted with topic separation and even 67% with separation by domain, while only 12% are needed (see Figure 15). This shows the security weakness of the hidden embedding approach. When a device participates in a tunnel, it can listen to all CFs of this tunnel and is permitted to send all CFs of the tunnel. A smaller number of filtered CFs eases attacks as fewer ECUs need to be compromised to gain control of the car. Even if gateways filter all unwanted and illegally sent messages it would still be possible to attack the NFs, e.g., by flooding the tunnels to delay time-critical flows. In the case of exposed embeddings, which fully isolates messages by SDN, attackers must compromise the exact sender of each CF on the Ethernet backbone to issue messages in this channel.

D. Benefits and Limitations in Characteristic Attack Scenarios

We investigate how robustly our setup of static flows augmented by ACL-controlled dynamic flows can isolate in-vehicle control flows and prevent unwanted traffic in case of example attacks. The attack surface of the IVN depends

heavily on which data flows are blocked or forwarded. We consider examples that affect data flows in the network to illustrate the impact of SDN flow control.

For characteristic attack scenarios we assume that the attacker succeeded in compromising the gateway so that attacks can be launched from there. Table V summarizes the results for our IVN with and without SDN access control. For the latter, the controller acts as a simple Ethernet switch.

TABLE V: Attacks launched from the online gateway in an unprotected environment and with SDN access control. A host scan searches for the existing 11 hosts, a port scan examines 4 open TCP and 8 open UDP ports on a host, and a TCP SYN flood of 1000 connections tries to impair the target.

Attack impact	**w/o access control**	**w/ access control**
Hosts discovered	11	11 (0 when arp is blocked)
Ports discovered	4 TCP, 8 UDP	0 TCP, 0 UDP
TCP SYN flood arrived	1000	0 (1000 at SDN controller)

Scans are the most common network attacks and used to gather information about the system. All eleven hosts can be discovered from the online gateway with *nmap* even with access control as long as ARP is not blocked by the ACL. Since hosts in the IVN are usually known, ARP can be blocked, which means that no hosts are discoverable. Scanning the ports of a host without access control, we discover 4 open TCP ports and 8 open UDP ports. With access control, no open ports can be discovered. This is true even for ports to which there is a connection, since the flow rules match source and destination ports and *nmap* uses the wrong source port. With elevated rights, however, the attacker could select the correct source port, which requires detailed knowledge about the vehicle and is even more difficult on embedded devices. Without scans, most attackers cannot perform further attacks.

As an example of a denial of service attack, we perform a TCP SYN flood attack on one of the discovered TCP ports. Without access control, all 1000 SYNs arrive at the target and the attack is successful. Since new source port and IP pairs are used for each SYN, they are detected and blocked by our ACL. Such attacks, however, are then forwarded by the switches to the SDN controller and could overload it. This is a known problem in SDN and protection mechanisms have been investigated in the past [60]. For example, most current SDN controllers can be distributed across multiple instances in standby to eliminate the single point of failure. In our case, the static flow rules in the switches are preserved even when the controller is unavailable, which guarantees safety-critical communication. Nevertheless, protection mechanisms for the controller should be further investigated in future work.

For targeted attacks, we assume that the adversary has detailed knowledge about the vehicle and attacks specific components. We analyze the impact of a replay attack, which is a targeted attack on multiple control flows. We recorded a 30 s trace of 10206 packets at ZC FL embedded exposed and hidden (per domain). Table VI shows the results for a replay from the online gateway. For conventional multicast, network device can react to unknown multicast addresses by either dropping or broadcasting the packet. Multicast groups that reach only devices on one switch are not registered on the other switch. With domain embedding, both switches are aware of all multicast groups, so all packets reach all destinations with both policies. Using per message embedding performs better because not every device joins all multicast groups. The original sender (ZC FL) does not receive packets with the drop policy because it is not a receiver of any of the multicast groups. The valid receivers, however, receive all packets from their registered multicast groups. With SDN access control, all packets are blocked since the ingress port connected to the online gateway is not a valid source of any of the flows. Herein lies the strength of our network-centric security approach. Regardless of the embedding, an attacker must control a legitimate sender of a CF in order to send it.

TABLE VI: Share of packets forwarded from a 30s replay (10206 packets total) recorded at ZC FL and played back from the online gateway. The results are shown for exposed and hidden embeddings each with SDN access control and conventional L2/L3 multicast. For the conventional forwarding unknown multicast groups are either dropped or broadcasted.

Embedding	**Dest.**	**Conventional multicast** (drop policy)	**Conventional multicast** (broadcast policy)	**w/ access control**
Per domain (hidden)	ZC FL	10206 (100%)	10206 (100%)	0
	ZC FR	10206 (100%)	10206 (100%)	0
	ZC RR	10206 (100%)	10206 (100%)	0
	ZC RL	10206 (100%)	10206 (100%)	0
Per message (exposed)	ZC FL	0	1903 (19%)	0
	ZC FR	7242 (71%)	10022 (98%)	0
	ZC RR	8617 (84%)	9525 (93%)	0
	ZC RL	4072 (40%)	4980 (49%)	0

In our IVN, there are no flows from the online gateway to the zonal controllers, so no direct driving commands can be sent. In future autonomous scenarios, larger ECUs will also use online services, e.g., for detailed up-to-date maps, and thus have communication paths to the online gateway. Targeted attacks, such as a, replay, packet flood, or forged information in an established network flow, are not prevented by our approach. Additional mechanisms such as anomaly detection can help detecting such attacks and countermeasures could be initiated by SDN, e.g., through reconfigurations.

E. Discussion

Our evaluation shows how SDN flow control can protect the IVN. In general, communication is drastically limited and attacks on unknown flows are detected and blocked. Regardless of the embedding strategy, Ethernet ECUs such as the online gateway are not allowed to send or receive control messages, which reduces the attack surface. This protects legacy ECUs that lack defense mechanisms from attacks by stronger ECUs as communication is forbidden between them.

On the other hand, targeted attacks that use flows already installed in forwarding devices cannot be prevented by our access control mechanism. Hidden embeddings lead to unintended receivers of critical CFs and permit their transmission from other network participants, posing a risk to safety and security. Exposed embedding allows the network to separate all

CFs, ensuring that only the original sending ZC is allowed to send the CF and that it only reaches the necessary destinations. This adds a layer of network security even in the case of a compromised gateway, but cannot prevent attacks from ECUs that are allowed to send certain CFs. Nevertheless, attacks are limited to the pre-installed and allowed flows and thus the attack surface is significantly reduced.

Our zone topology with SDN-enforced message separation is also more secure than the original network architecture of our production vehicle. In a pure CAN bus architecture, traffic on the buses cannot be controlled. All ECUs connected to a domain bus can send any CF and receive any message on the bus. A compromised ECU can attack all other ECUs in its domain. Gateways can use the CAN IDs to filter messages to be forwarded, but cannot verify the correct sender of a CF. The zone topology splits the CAN domain buses so that fewer ECUs are connected to a physical bus. The gateways can thus filter messages and fewer ECUs are completely unprotected. If each ECU were directly connected to the zone controller, the correct sender could be verified. Combined with separation of messages by SDN, this could offer perfect separation of CFs and thus a robust, trustworthy communications backbone.

In all attack scenarios, the SDN controller must be well protected because it could be the target of attacks itself. In our case, the static configuration protects the safety-critical traffic in the IVN, since it cannot be changed by the controller. Future work could perform a risk analysis for attacks on the SDN controller in vehicles.

VII. Conclusion and Outlook

In this work, we investigated the integration of TSN with SDN for improving network security in Ethernet-based IVNs. We presented a TSSDN switching architecture that harmonizes the functions of TSN and SDN. At its core, our approach implements SDN flow control for simultaneous asynchronous and synchronous real-time as well as best-effort traffic in vehicles. We could show how time-sensitive flows can be reserved via OpenFlow, and how TDMA schedules can be reconfigured at runtime without sacrificing the real-time capabilities of TSN.

Targeting at network-level security by isolation, we comparatively evaluated three strategies for mapping control flows into a software-defined Ethernet backbone of a real-world IVN, which we transformed into a realistic software-defined Ethernet topology. Our analysis revealed that network security and performance can be largely improved by exposing control flow properties in standard network header fields which are processed by forwarding devices. Embeddings that are hidden within application layer protocols lead to significant oversupply of control flows, which opens the attack surface.

Future work shall analyze the impact of network-wide schedule reconfiguration strategies on real-time traffic in different scenarios. A mechanism for TSN senders of scheduled traffic is needed to exchange requirements with the SDN controller and announce the start of a transmission. Additional network-level intelligence can further improve in-vehicle security, as there are still unused options applicable as security guards based on SDN monitoring and control.

References

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

[2] S. Brunner *et al.*, "Automotive E/E-Architecture Enhancements by Usage of Ethernet TSN," in *2017 13th WS on Intelligent Solutions in Embedded Systems (WISES)*. IEEE, 2017, pp. 9–13.

[3] IEEE 802.1 Working Group, "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," Standard IEEE 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014), Jul. 2018.

[4] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 38, pp. 69–74, 2008.

[5] K. Halba *et al.*, "Robust Safety for Autonomous Vehicles through Reconfigurable Networking," in *2nd International WS on Safe Control of Autonomous Vehicles*, ser. Electronic Proc. in Theoretical Computer Science, vol. 269. Open Publishing Association, 2018, pp. 48–58.

[6] T. Häckel *et al.*, "Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*. IEEE, Apr. 2019, pp. 1–5.

[7] M. Haeberle *et al.*, "Softwarization of Automotive E/E Architectures: A Software-Defined Networking Approach," in *2020 IEEE Vehicular Networking Conference (VNC)*. IEEE, Dec. 2020, pp. 1–8.

[8] N. G. Nayak *et al.*, "Time-sensitive Software-defined Network (TSSDN) for Real-time Applications," in *24th International Conf. on Real-Time Networks and Systems*, ser. RTNS '16. ACM, 2016, pp. 193–202.

[9] P. Mundhenk, *Security for Automotive Electrical/Electronic (E/E) Architectures*. Göttingen: Cuvillier, Aug. 2017.

[10] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," *Black Hat USA*, vol. 2015, p. 91, 2015.

[11] S. Shin *et al.*, "Enhancing Network Security through Software Defined Networking (SDN)," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, Aug. 2016.

[12] O. Yurekten and M. Demirci, "SDN-based cyber defense: A survey," *Future Gen. Computer Systems*, vol. 115, pp. 126–149, Feb. 2021.

[13] T. Häckel *et al.*, "Strategies for Integrating Controls Flows in Software-Defined In-Vehicle Networks and Their Impact on Network Security," in *2020 IEEE Vehicular Networking Conf. (VNC)*. IEEE, Dec. 2020.

[14] M. Dibaei *et al.*, "Attacks and defences on intelligent connected vehicles: a survey," *Digital Com. and Networks*, vol. 6, pp. 399–421, Nov. 2020.

[15] AUTOSAR, "SOME/IP Protocol Specification," Std. 696, Nov. 2021.

[16] A. Kampmann *et al.*, "A Dynamic Service-Oriented Software Architecture for Highly Automated Vehicles," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2101–2108.

[17] L. Lo Bello and W. Steiner, "A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems," *Proceedings of the IEEE*, vol. 107, pp. 1094–1120, Jun. 2019.

[18] T. Steinbach *et al.*, "Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)," in *2012 IEEE Vehicular Technology Conf. (VTC Fall)*. IEEE, Sep. 2012.

[19] V. Gavriluţ *et al.*, "AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN," *IEEE Access*, vol. 6, pp. 75 229–75 243, Nov. 2018.

[20] A. A. Syed *et al.*, "MIP-based Joint Scheduling and Routing with Load Balancing for TSN based In-vehicle Networks," in *2020 IEEE Vehicular Networking Conference (VNC)*. IEEE, Dec. 2020.

[21] R. Enns *et al.*, "Network Configuration Protocol (NETCONF)," IETF, RFC 6241, June 2011.

[22] D. Kreutz *et al.*, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, pp. 14–76, Jan. 2015.

[23] Open Networking Foundation, "OpenFlow Switch Specification," ONF, Standard ONF TS-025, 2015.

[24] D. Thiele and R. Ernst, "Formal Analysis Based Evaluation of Software Defined Networking for Time-Sensitive Ethernet," in *2016 Design, Automation & Test in Europe (DATE)*. IEEE, Mar. 2016, pp. 31–36.

[25] N. G. Nayak *et al.*, "Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks," *IEEE Transactions on Industrial Informatics*, vol. 14, pp. 2066–2075, 2018.

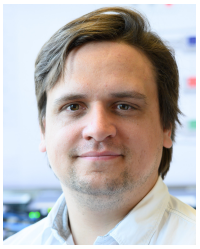
[26] T. Steinbach *et al.*, "Beware of the Hidden! How Cross-traffic Affects Quality Assurances of Competing Real-time Ethernet Standards for In-Car Communication," in *2015 IEEE LCN*, Oct. 2015, pp. 1–9.

[27] T. Gerhard *et al.*, "Software-defined Flow Reservation: Configuring IEEE 802.1Q Time-Sensitive Networks by the Use of Software-Defined Networking," in *2019 24th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*. IEEE, Sep. 2019, pp. 216–223.

[28] S. Nam *et al.*, "Simplified Stream Reservation Protocol over Software-Defined Networks for In-vehicle Time-Sensitive Networking," *IEEE Access*, pp. 1–12, Jun. 2021.

[29] J. Cui *et al.*, "Transaction-Based Flow Rule Conflict Detection and Resolution in SDN," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, Jul. 2018, pp. 1–9.
[30] M. Curic *et al.*, "Transactional Network Updates in SDN," in *2018 Eur. Conf. on Networks and Communications (EuCNC)*. IEEE, Jun. 2018.
[31] R. Rotermund *et al.*, "Requirements Analysis and Performance Evaluation of SDN Controllers for Automotive Use Cases," in *2020 IEEE Vehicular Networking Conference (VNC)*. IEEE, Dec. 2020.
[32] S. Checkoway *et al.*, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *20th USENIX Security Symposium*, vol. 4. USENIX Association, Aug. 2011, pp. 77–92.
[33] L. Wang *et al.*, "k-Zero Day Safety: A Network Security Metric for Measuring the Risk of Unknown Vulnerabilities," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, pp. 30–44, Jan. 2014.
[34] A. Ruddle *et al.*, "Security Requirements For Automotive On-Board Networks Based On Dark-Side Scenarios," *Evita Deliverable 2.3*, 2009.
[35] J.-P. Monteuuis *et al.*, "SARA: Security Automotive Risk Analysis Method," in *4th ACM WS on Cyber-Physical System Security*, ser. CPSS '18. ACM, 2018, pp. 3–14.
[36] S. Longari *et al.*, "A Secure-by-Design Framework for Automotive On-board Network Risk Analysis," in *2019 IEEE Vehicular Networking Conference (VNC)*. IEEE, Dec. 2019.
[37] V. L. L. Thing and J. Wu, "Autonomous Vehicle Security: A Taxonomy of Attacks and Defences," in *2016 IEEE Int. Conference on iThings and GreenCom and CPSCom and SmartData*. IEEE, Dec. 2016.
[38] M. D. Pesé *et al.*, "Hardware/Software Co-Design of an Automotive Embedded Firewall," in *SAE Technical Paper*. SAE Int., Mar. 2017.
[39] M. Rumez *et al.*, "Integration of Attribute-based Access Control into Automotive Architectures," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, Jun. 2019.
[40] S. Seifert and R. Obermaisser, "Secure Automotive Gateway - Secure Communication for Future Cars," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 2014, pp. 213–220.
[41] G. K. Rajbahadur *et al.*, "A Survey of Anomaly Detection for Connected Vehicle Cybersecurity and Safety," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, Jun. 2018.
[42] L. Yang *et al.*, "Tree-Based Intelligent Intrusion Detection System in Internet of Vehicles," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.
[43] P. Meyer *et al.*, "Network Anomaly Detection in Cars based on Time-Sensitive Ingress Control," in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*. IEEE, Nov. 2020, pp. 1–5.
[44] P. Waszecki *et al.*, "Automotive Electrical and Electronic Architecture Security via Distributed In-Vehicle Traffic Monitoring," *IEEE TCAD*, vol. 36, pp. 1790–1803, Nov. 2017.
[45] F. Langer *et al.*, "Establishing an Automotive Cyber Defense Center," in *17th escar Europe : embedded security in cars*, 2019.
[46] Q. Hu and F. Luo, "Review of Secure Communication Approaches for In-Vehicle Network," *Int. J. Auto. Tech.*, vol. 19, pp. 879–894, Sep. 2018.
[47] S. Fassak *et al.*, "A secure protocol for session keys establishment between ECUs in the CAN bus," in *2017 International Conf. on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, Nov. 2017.
[48] D. Püllen *et al.*, "Securing FlexRay-Based In-Vehicle Networks," *Microprocessors and Microsystems*, p. 103144, Jun. 2020.
[49] IEEE, "IEEE Standard for Local and metropolitan area networks-Media Access Control (MAC) Security," Std. IEEE 802.1AE-2018, Dec. 2018.
[50] M. Khodaei *et al.*, "SECMACE: Scalable and Robust Identity and Credential Management Infrastructure in Vehicular Communication Systems," *IEEE T-ITS*, vol. 19, pp. 1430–1444, May 2018.
[51] M. Rumez *et al.*, "An Overview of Automotive Service-Oriented Architectures and Implications for Security Countermeasures," *IEEE Access*, vol. 8, pp. 221 852–221 870, 2020.
[52] L. Leonardi *et al.*, "Bandwidth partitioning for Time-Sensitive Networking flows in automotive communications," *IEEE Communications Letters*, pp. 3258–3261, 2021.
[53] A. Kern *et al.*, "Gateway Strategies for Embedding of Automotive CAN-Frames into Ethernet-Packets and Vice Versa," in *Architecture of Computing Systems - ARCS 2011*. Springer, 2011, pp. 259–270.
[54] OpenSim Ltd., "OMNeT++ Discrete Event Simulator." [Online]. Available: https://omnetpp.org/
[55] T. Häckel *et al.*, "SDN4CoRE: A Simulation Model for Software-Defined Networking for Communication over Real-Time Ethernet," in *6th International OMNeT++ Community Summit 2019*, ser. EPiC Series in Computing, vol. 66. EasyChair, Dec. 2019, pp. 24–31.
[56] OpenSim Ltd., "INET Framework." [Online]. Available: https://inet.omnetpp.org/
[57] D. Klein and M. Jarschel, "An OpenFlow Extension for the OMNeT++ INET Framework," in *6th International ICST Conference on Simulation Tools and Techniques*, ser. SimuTools '13. ICST, 2013, pp. 322–329.
[58] P. Meyer *et al.*, "Simulation of Mixed Critical In-vehicular Networks," in *Recent Advances in Netw. Simulation*. Springer, 2019, pp. 317–345.
[59] P. Meyer *et al.*, "Extending IEEE 802.1 AVB with Time-triggered Scheduling: A Simulation Study of the Coexistence of Synchronous and Asynchronous Traffic," in *2013 IEEE VNC*. Dec. 2013, pp. 47–54.
[60] T. Han *et al.*, "A comprehensive survey of security threats and their mitigation techniques for next-generation SDN controllers," *Concurrency and Computation*, vol. 32:e5300, pp. 1–21, 2020.

Timo Häckel received his M.Sc. degree in computer science from the Hamburg University of Applied Sciences (HAW), Hamburg, Germany, in 2018, where he is currently pursuing the Ph.D. degree with the Internet Technologies Research Group. His particular research interest is the security of time-sensitive vehicular networks, which he is exploring within the German research project SecVI – Security for Vehicular Information. Timo Häckel is also part of the Communication over Real-Time Ethernet research group at HAW Hamburg, where he develops and maintains open source frameworks for the OMNeT++ simulator including the CoRE4INET and SDN4CoRE framework.



Philipp Meyer is Ph.D. student and research assistant in the Communication over Real-Time Ethernet (CoRE) Research Group at the Hamburg University of Applied Sciences (HAW), Hamburg, Germany. As part of the CoRE Research Group, he started 2012 with researching on real-time Ethernet technologies and their security. He received his B.Sc. (2013) and M.Sc. (2018) in computer science in this context. Furthermore, he pursued his interests as part of the research projects Realtime Ethernet Backbone for Cars (RECBAR) and X-Check. Currently, he is exploring in-car communication security as part of the research project Security for Vehicular Information (SecVI). Philipp also develops and maintains the CoRE open-source simulation environment and frameworks for OMNeT++.

Franz Korf is professor of Embedded Systems at Hamburg University of Applied Sciences (HAW), where he heads the Communication over Real-Time Ethernet research group. He studied computer science at RWTH Aachen University. At the chair of W. Damm at the University of Oldenburg, he received his doctorate in the field of system-level synthesis tools. Before joining HAW Hamburg in 2004, he headed the OEM development of server systems at Fujitsu Siemens Computers. At HAW Hamburg, Franz was responsible for various R & D projects in the areas of real-time Ethernet architectures and embedded systems.

Thomas C. Schmidt is professor of Computer Networks and Internet Technologies at Hamburg University of Applied Sciences (HAW), where he heads the Internet Technologies research group (iNET). He studied mathematics, physics and German literature at Freie Universitaet Berlin and University of Maryland, and received his Ph.D. from FU Berlin in 1993. Since then he has continuously conducted numerous national and international research projects. His continued interests lie in the development, measurement, and analysis of large-scale distributed systems like the Internet. He serves as co-editor and technical expert in many occasions and is actively involved in the work of IETF and IRTF. Thomas is a co-founder of several large open source projects and coordinator of the community developing RIOT—the friendly Operating System for the IoT.