

Article

A Robust Multi-Port Network Interface Architecture with Real-Time CRC-Based Fault Recovery for In-Vehicle Communication Networks

Sungju Lee ¹ , Sungwook Yu ^{2,*}  and Taikyeong Jeong ^{3,4,*}¹ Department of Software, Sangmyung University, Chunan 31066, Republic of Korea; peacfeel@smu.ac.kr² School of Electrical and Electronics Engineering, Chung-Ang University, Seoul 06974, Republic of Korea³ School of Artificial Intelligence Convergence, Hallym University, Chuncheon 24252, Republic of Korea⁴ Department of Psychiatry, College of Medicine, Hallym University, Chuncheon 24252, Republic of Korea

* Correspondence: sungwook@cau.ac.kr (S.Y.); ttjeong@hallym.ac.kr (T.J.)

Abstract

As the automotive industry continues to evolve rapidly, there is a growing demand for high-throughput reliable communication systems within vehicles. This paper presents the implementation and verification of a fault-tolerant Ethernet-based communication protocol tailored for automotive applications operating at 1 Gbps and above. The proposed system introduces a multi-port Network Interface Controller (NIC) architecture that supports real-time communication and robust fault handling. To ensure adaptability across various in-vehicle network (IVN) scenarios, the system allows for configurable packet sizes and transmission rates and supports diverse data formats. The architecture integrates cyclic redundancy check (CRC)-based error detection, real-time recovery mechanisms, and file-driven data injection techniques. Functional validation is performed using Verilog HDL simulations, demonstrating deterministic timing behavior, modular scalability, and resilience under fault injection. This paper presents a fault-tolerant Network Interface Controller (NIC), architecture incorporating CRC-based error detection, real-time recovery logic, and file-driven data injection. The system is verified through Verilog HDL simulation, demonstrating correct timing behavior, modular scalability, and robustness against injected transmission faults. Compared to conventional dual-port NICs, the proposed quad-port architecture demonstrates superior scalability and error tolerance under injected fault conditions. Experimental results confirm that the proposed NIC architecture achieves stable multi-port communication under embedded automotive environments. This study further introduces a novel quad-port NIC with an integrated fault injection algorithm and evaluates its performance in terms of error tolerance.

Keywords: fault-tolerant; automotive ethernet; network interface controller (NIC); fault tolerance; CRC; Verilog HDL; in-vehicle network (IVN); real-time communication



Academic Editor: Xiaozheng Jin

Received: 1 July 2025

Revised: 2 August 2025

Accepted: 3 August 2025

Published: 7 August 2025

Citation: Lee, S.; Yu, S.; Jeong, T. A Robust Multi-Port Network Interface Architecture with Real-Time CRC-Based Fault Recovery for In-Vehicle Communication Networks. *Actuators* **2025**, *14*, 391. <https://doi.org/10.3390/act14080391>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In the vehicle communication system, research on fault tolerance in the large-capacity communication network is rapidly increasing due to the increase in the amount of vehicle data transmission along with the bandwidth demand for large-capacity data transmission. Although various studies are underway for large-capacity data transmission [1], the research should be considered in the environments with Giga-band or higher bandwidth communication. Meanwhile, modern vehicles are increasingly equipped with systems

such as brake and stability control, lane departure warning systems (LDWS), in-vehicle infotainment systems (IVI) offering various convenience features via electronic control units (ECUs), and safety enhancement functions such as parking assistance systems (PAS) [2]. Remote vehicle diagnosis, the emergency rescue vehicle, such as the required size of the data communication service, is also increasing. As a result, many companies are subject to a vehicle with synchronous Ethernet-based networks, which are applied to the in-vehicle network (*herein after called INV*) pertaining to the most important standard [2]. The global automotive industry, the automotive software platform (e.g., AutoSAR [3], GINI-V [4]), and synchronous Ethernet with the same standard for building a collaborative research environment, starting with an open system for automotive electronics, aim to standardize the interface using very active research [5].

In particular, image-based obstacle recognition systems, and in a variety of applications such as automotive black box processing systems, require large amounts of information around the vehicle and are the standardization of the network backbone to be prepared. Especially, RAPIEnet [6], an Ethernet standard that has established itself as an international standard for industrial Ethernet, is widely used in the development and production of vehicle safety systems, including in applications approved as real-time automation protocols for Ethernet [7].

Ethernet has emerged as a critical technology for enabling synchronous communication between electronic control units (ECUs), maintaining a consistent phase relationship between transmission and reception to ensure reliable interactions among multiple devices. With the rapid surge in user demand, the required network bandwidth is increasing at an unprecedented rate [8]. Consequently, existing vehicular communication technologies such as Controller Area Network (CAN) and FlexRay [9] are being re-evaluated worldwide, including through experiments involving novel types of soft errors, in efforts to overcome their limitations in high-capacity data transmission.

Figure 1 is a hierarchical explanation of the in-vehicle communication structure. It refers to the overall configuration of the communication system inside and outside the vehicle, designed based on the backbone network for high-speed communication called synchronous Ethernet (Automotive Intelligent Network) at the top. In particular, it supports large-capacity data transmission and time-synchronized information exchange between major systems inside the vehicle. At the bottom, the Infotainment Bus and Control BUS (e.g., CAN, FlexRay) are explained separately. The Infotainment Bus processes user-centered infotainment functions such as car navigation, audio, and media, and the Control BUS stably processes the real-time data of the control system (braking, steering, engine, etc.). If this integrated network architecture is actually tested, it will be utilized as a core basic technology for implementing intelligent systems in vehicles.

In this paper, we first present a theoretical framework for analyzing a multi-port fault tolerance mechanism. The proposed approach addresses the increasing complexity of in-vehicle networks (IVNs) by enabling robust communication among multiple nodes, even under fault injection scenarios. This analysis serves as a foundational step toward the development of reliable automotive communication protocols. Also, we develop a scalable performance evaluation framework based on a packet-level (i.e., data-level) injection system. This framework supports flexible testing across various data rates and communication scenarios, making it suitable for a broad range of automotive applications. The proposed architecture enables modular scalability and repeatable testing procedures, providing a practical means for validating both the functional reliability and performance efficiency of embedded in-vehicle network systems. Finally, we design and implement a Network Interface Controller (NIC) at the hardware-level using Verilog Hardware Description Language (HDL). The hardware architecture enables protocol-level verification

through simulation, ensuring accurate timing behavior and real-time fault recovery. This contribution bridges the gap between theoretical modeling and practical implementation, offering a viable blueprint for the development of fault-tolerant NIC architectures in automotive environments.

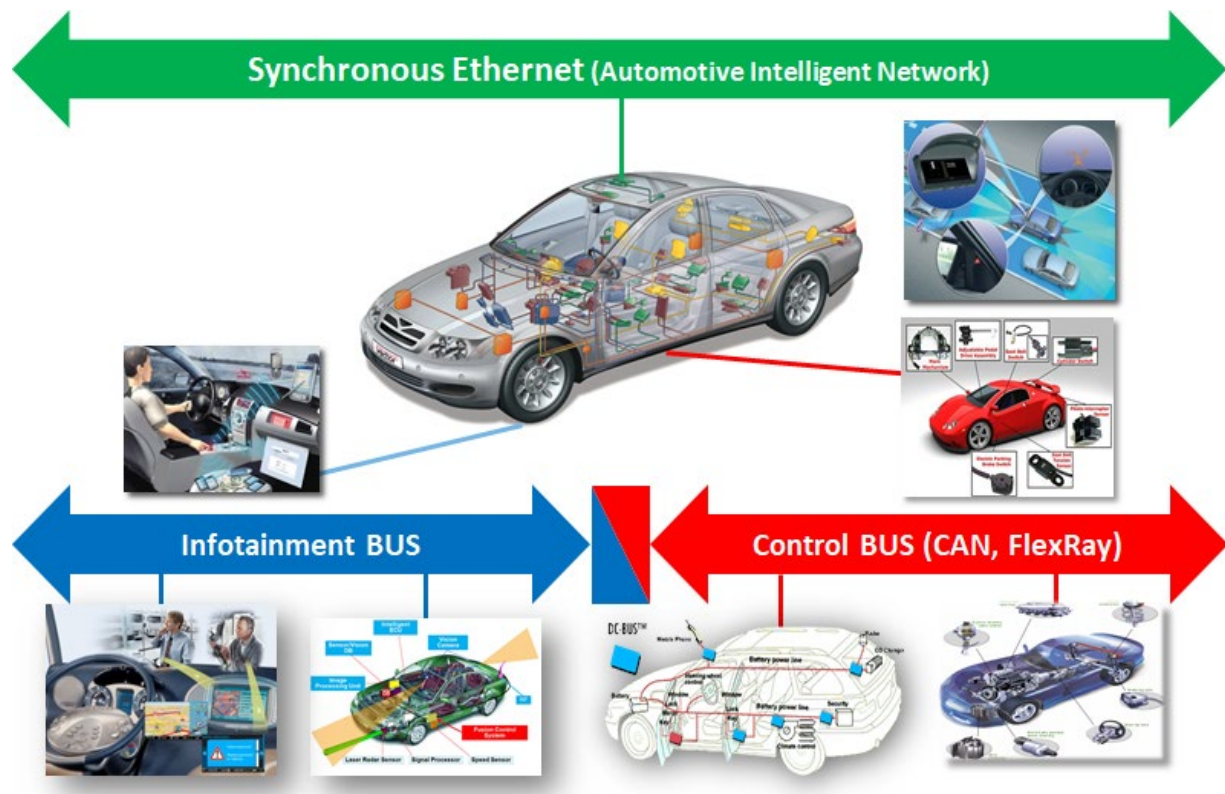


Figure 1. The vehicle communication model showing integration of synchronous Ethernet, infotainment data paths, and control-level communication over Control Bus (CAN and FlexRay) [10].

A preliminary research paper suggested that, in the case of a dual-port hardware redundancy system with four or more grants from the side using the NIC port to control the algorithm, it proves more unstable and is mathematically very difficult [11–13]. The differences and contributions compared to existing methods are summarized as follows.

- We design and implement a fault-tolerant Ethernet-based NIC architecture for in-vehicle networks (IVNs), supporting 1 Gbps throughput and multi-port configurations (≥ 4 ports), whereas prior works have primarily focused on dual-port redundancy.
- We propose a novel fault injection and recovery algorithm that integrates CRC-based error detection with real-time correction logic, enabling fast recovery ($< 1 \mu s$) from injected transmission faults.
- We provide a comparative evaluation against conventional protocols (e.g., CAN, FlexRay, and commercial NICs such as RAPIenet and BroadR-Reach), highlighting the proposed system's superior scalability, deterministic timing, and robustness under fault conditions.

In particular, we evaluate the communication performance of the IVN from a system-wide fault tolerance perspective by utilizing a commercial automotive physical layer solution BCM89810, a single monolithic CMOS chip [14,15].

The overall structure of this paper is as follows. Section 2 describes the background and related research. Section 3 presents a theoretical discussion on nodes, ports, and connections of a multi-port NIC card. Section 4 discusses the methodology for verifying the

system with a fault-tolerant algorithm. Section 5 presents a network-based performance evaluation by setting up a test environment, and Section 6 presents verification results using different data types for performance evaluation. Section 7 presents performance verification of fault tolerance through hardware design using an actual HDL design method. Finally, Section 8 presents our conclusions.

2. Background and Related Work

Since the early 1960s, engineers have recognized the importance of fault injection in the design of microprocessor-based electronic systems. However, at the time, simulation models were often implemented late in the design process, immediately before final validation, resulting in limited effectiveness in verifying timing-related issues [16]. Early relatively simple embedded systems could be verified using basic simulations. However, as embedded systems increased in complexity and functionality, the potential impact of deploying defective products grew significantly, leading to increased costs and risks associated with failure. Consequently, it became essential to test system reliability through software simulations before hardware production and board-level integration [17].

Fault injection methodologies are generally classified into three categories: (i) hardware-based approaches, (ii) software-implemented techniques, and (iii) simulation-based methods [18,19]. Simulation-based techniques are typically used in early design stages, whereas hardware fault injection is more commonly applied after manufacturing physical prototypes.

Design engineers frequently utilize high-level modeling techniques such as Petri nets [20], queuing models, and Dependability Parameter Estimation (DPE) to evaluate fault tolerance. However, to achieve precise and meaningful results, it is often necessary to refine these models at a lower level of abstraction [21]. System reliability is increasingly being verified through software-based simulations as a prerequisite for real-world deployment [22].

Although several global companies have already developed and commercialized Ethernet and synchronous Ethernet (SyncE) protocols and PHYs, research and development in the automotive domain remain incomplete. Notably, comprehensive testing methodologies that integrate both hardware and software components are still lacking [23]. This limitation highlights the urgent need for holistic test frameworks that address the unique fault tolerance requirements of vehicular networks.

In this context, the present study proposes a high-level fault-tolerant algorithm and validates its applicability within a synchronous Ethernet-based in-vehicle communication system. Prior research on fault-tolerant communication and SyncE integration is also reviewed to ensure the algorithm's compatibility and effectiveness under realistic network conditions [23].

Synchronous Ethernet systems are inherently vulnerable to physical-layer failures such as line short circuits, which can severely compromise reliability. To address this, redundant fault-tolerant Ethernet architectures have been proposed, aiming to minimize the probability of communication loss by leveraging partial redundancy [24]. Traditional Ethernet systems transmit packets over a shared medium using a best-effort approach. While this method provides high bandwidth utilization, it does not support deterministic or real-time behavior, which is crucial for automotive applications [25].

Achieving accurate time synchronization is critical in distributed systems that require coordinated data exchange. The IEEE 1588 Precision Time Protocol (PTP) has been proposed as the foundational technology for synchronized communication in such systems [26,27]. Current IEEE standardization efforts on synchronous Ethernet are gaining attention due

to their potential to overcome the limitations of asynchronous Ethernet, particularly by enabling deterministic quality-of-service (QoS) guarantees [28].

Recently, as the complexity of in-vehicle communication systems increases, various IEEE-based communication technologies are being introduced. In particular, the IEEE 802.3ch-2020 standard supports high-speed Ethernet communication of 2.5 Gb/s, 5 Gb/s, and 10 Gb/s in a vehicle environment, enabling real-time transmission of high-resolution sensor data and high-speed control signals [27,29].

In addition, the IEEE 802.1AS-2020 standard is a standard for time synchronization of in-vehicle networks, providing precise time synchronization at the $\leq 1 \mu\text{s}$ threshold level to meet the requirements of real-time control and multimedia applications [30].

Since the introduction of such high-speed communication and precise time synchronization technologies directly affects the capacity and performance of in-vehicle communication networks, it is essential to verify performance considering communication capacity from the design stage. This ensures the stability and reliability of the system and guarantees the scalability and compatibility of future vehicle communication systems. Based on these technological advances, it is important to actively utilize IEEE standards in the design and verification process of vehicle communication systems, which will play a key role in the development of autonomous driving and connected car technologies in the future.

In the automotive domain, manufacturers face numerous challenges in analyzing and controlling internal data flow. During the development of new hardware and software models, system safety must be ensured through rigorous analysis and validation of underlying data structures. In particular, safety-critical applications demand verified system behavior before deployment. Therefore, simplified yet efficient transport protocols capable of handling real-world data within the constraints of embedded controllers must be designed. By incorporating fault tolerance into these communication frameworks, developers can improve reliability while expanding the system's applicability across a broader range of automotive and industrial use cases [31]. Existing in-vehicle network (IVN) standards, such as CAN and FlexRay, have been widely deployed in automotive control systems due to their reliability and cost efficiency. However, their limited bandwidth (CAN $\leq 1 \text{ Mbps}$, FlexRay $\leq 10 \text{ Mbps}$) cannot accommodate the growing demand for high-throughput data transmission required by modern sensor and infotainment systems. BroadR-Reach (IEEE 802.3bw, 100BASE-T1) addresses wiring complexity by enabling 100 Mbps over a single UTP cable, but it does not natively provide deterministic or fault-tolerant communication. RAPIEnet, originally developed for industrial Ethernet, supports redundancy and Ring topologies with recovery times $< 1 \text{ ms}$, making it suitable for safety systems, but its deployment in automotive environments remains limited. Synchronous Ethernet (SyncE) ensures clock-level synchronization but requires additional protocols for error tolerance.

In contrast, the proposed NIC architecture integrates CRC-based error detection, real-time recovery logic ($< 1 \mu\text{s}$), and multi-port scalability (≥ 4 ports). This design ensures deterministic timing, robust fault tolerance, and scalability to 1 Gbps, surpassing existing IVN technologies. A comparative summary is presented in Table 1.

As shown in Table 1, the proposed NIC outperforms conventional IVN technologies by combining the scalability of Ethernet with built-in fault-tolerant mechanisms. Unlike BroadR-Reach, which prioritizes cost and cabling efficiency, the proposed system achieves both high bandwidth and deterministic timing. Compared to RAPIEnet and SyncE, the proposed NIC provides faster recovery from injected faults ($\leq 1 \mu\text{s}$) and ensures robust operation under multi-port conditions, making it particularly suitable for safety-critical and real-time automotive applications.

Table 1. Comparison of automotive communication technology.

Technology	Data Rate	Fault Tolerance	Determinism (Timing)	Scalability	Remark
CAN	≤ 1 Mbps	High (via arbitration)	Moderate (non-deterministic under load)	Low	Legacy, widely used
FlexRay	≤ 10 Mbps	Moderate (dual-channel)	Deterministic TDMA	Low–moderate	Used in safety-critical ECUs
BroadR-Reach	100 Mbps (IEEE 802.3bw)	Low (no native redundancy)	Deterministic TDMA	Moderate	Reduces cabling complexity
RAPiEnet	100 Mbps–1 Gbps	High (redundant ring, < 1 ms recovery)	Moderate–high	Moderate	Industrial-grade Ethernet
SynE	1 Gbps	Low (requires additional mechanisms)	High (clock-level sync)	High	Strong synchronization, but limited in automotive fault handling
Proposed NIC	≥ 1 Gbps	Very high (CRC-based + recovery < 1 μ s)	High (clock-level sync)	High (multi-port ≥ 4)	Automotive-focused, scalable, robust

3. Theoretical Analysis

In the context of modern high-performance computing systems and data-intensive embedded architectures, the design of scalable and efficient NICs plays a pivotal role in determining the overall communication performance, fault tolerance, and expandability of the network subsystem. To facilitate the theoretical foundation for such NIC architectures, we define a formal model of NIC-port configurations and interconnect structures that serve as the basis for performance analysis and architectural decisions.

Figure 2 illustrates the inter-node communication structure, where each node is modeled as a multi-port module. Using the formal port-to-port mapping $C_{i,j}^{(k)}$, we define a scalable connection model capable of supporting redundancy, bandwidth aggregation, or failover routing, depending on the topology constraints and application context. This architecture enhances fault resilience and improves system availability under link or port failure conditions.

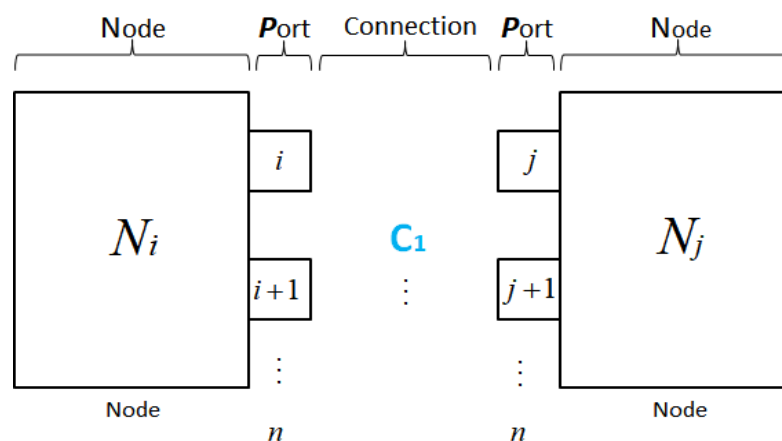


Figure 2. An abstract representation of a connection between two communication nodes, N_i and N_j , each consisting of multiple ports. Ports i to n on node N_i interact with ports j to n on node N_j through a connection channel C_1 . This model is used to represent port-to-port mappings in network interface architectures supporting multi-channel communication. The structure allows for scalable configuration for high-redundancy or parallel transmission paths.

In the proposed multi-port communication model, each node N_i is equipped with n ports, indexed from i to $i + n - 1$. The connection between two nodes N_i and N_j is established via a dedicated port-to-port mapping, denoted as follows:

$$C_{i,j}^{(k)} = P_i^{(k)} \longleftrightarrow P_j^{(k)} \quad (1)$$

where $P_i^{(k)}$ and $P_j^{(k)}$ represent the k -th port of nodes N_i and N_j , respectively, and $C_{i,j}^{(k)}$ denotes the k -th connection channel between the two nodes.

The complete connection set $C_{i,j}$ between N_i and N_j can be defined as follows.

$$C_{i,j} = \{C_{i,j}^{(k)} \mid k = 1, 2, \dots, m\} \quad (2)$$

Therefore, we can define the relationship between these two relations again as a mathematical formula. Let each node N_i be equipped with n ports, denoted as $P_i = \{P_{i1}, P_{i2}, \dots, P_{in}\}$. A directed communication link between port P_{ia} on node N_i and port P_{jb} on node N_j is defined as $C_{i,j}(a,b) = P_{ia} \rightarrow P_{jb}$. The complete set of possible connections between nodes is, therefore, $C_{i,j} = \{C_{i,j}(a,b) \mid a,b \in [1,n]\}$.

This summation is generalizable to NICs with heterogeneous designs, allowing for the inclusion of devices with different bandwidth capacities, signaling standards, or functional roles.

Additionally, this again raises the need to clearly examine the mathematical model and analysis from the perspective of multiple multi-ports (P) that a one node (N) has. Let us examine this again from a connection (C) oriented interpretation rather than a port-oriented interpretation.

Figure 3 depicts a connection-oriented communication model between two nodes, N_i and N_j , each composed of multiple ports. Rather than focusing solely on individual ports, the model highlights the logical connections $C_x \in C_{\{i,j\}}$ that represent directed data paths between source and destination ports. This abstraction enables a modular understanding of how data flow across the network layer, especially in systems with multi-port Network Interface Controllers (NICs).

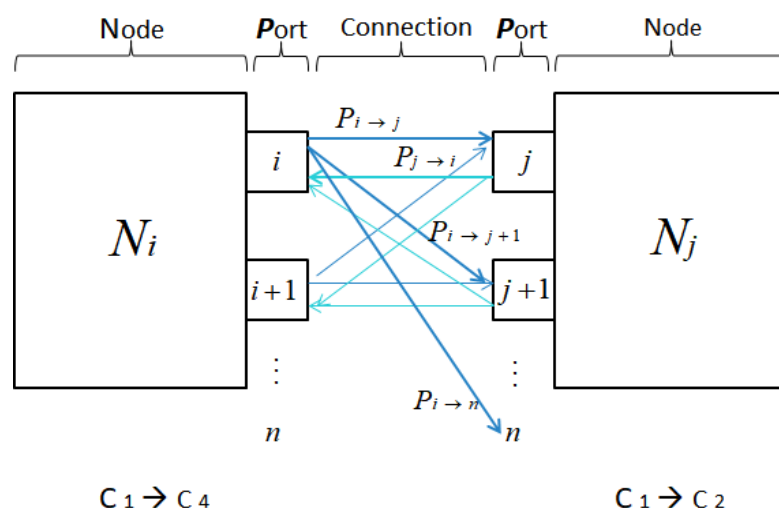


Figure 3. A connection-oriented model between nodes N_i and N_j , where multiple directed links represent logical communication paths between specific ports. Unlike a port-centric view, the diagram emphasizes the structure and semantics of the connection set $\{C_1, C_2, \dots, C_k\}$, where each C_x defines a distinct mapping from source port P_i to target port P_j . This representation facilitates a scalable analysis of network routing, redundancy, and fault-tolerant communication mechanisms across distributed systems.

The bold dark-blue arrows represent active and valid one-way transmission paths (e.g., $P_i \rightarrow P_j$, $P_i \rightarrow P_{[j+1]}$, or $P_i \rightarrow P_n$). These connections are currently engaged in data transmission and are monitored for integrity and consistency. Conversely, the light-blue arrows indicate potential or standby links (e.g., $P_j \rightarrow P_i$), which are either inactive or in passive listening mode, serving redundancy, load balancing, or fault recovery purposes.

Each connection C_x is defined formally as follows.

$$C_x = (P_s, P_d), \text{ where } P_s \in N_i, P_d \in N_j \quad (3)$$

The complete connection set between the two nodes is given by the following:

$$C_{\{i,j\}} = \{C_1, C_2, \dots, C_x \mid C_x = (P_i^{(x)}, P_j^{(x)})\} \quad (4)$$

where $k \leq n^2$ depending on whether the network uses full-mesh or selective connections.

The notation $C_1 \Rightarrow C_4$ and $C_1 \Rightarrow C_2$ annotated below the figure suggests the dynamic redirection or logical mapping of connection identifiers based on context or rerouting policies. This highlights the flexibility of the communication layer to adapt connection roles or endpoints under dynamic runtime conditions.

To further describe the port-to-port relationship, consider two arbitrary ports: P_i^a , the a -th port on NIC i , and P_j^b , the b -th port on NIC j . These two ports may be physically or logically connected via a communication channel $C_{\{i,j\}}^{\{(a,b)\}}$, defined as follows.

$$C_{\{i,j\}}^{\{(a,b)\}} = P_i^a \rightarrow P_j^b \quad (5)$$

This directional connection indicates that port a on NIC i is transmitting to port b on NIC j . In symmetric or full-duplex systems, the reverse link may also be assumed.

$$C_{\{i,j\}}^{\{(a,b)\}} = P_j^b \rightarrow P_i^a \quad (6)$$

The nature of the communication between P_i^a and P_j^b is significant in architectural design. For instance, one-to-one deterministic mappings are typically used in latency-critical embedded systems, whereas shared multi-point mappings may be employed in data center fabrics to support flexible routing. The number of such connections, their topological arrangement, and their symmetry properties directly influence routing table size, congestion management, and fault containment strategies.

For a system with n NICs, each with n ports, the full set of pairwise inter-NIC port connections forms a complete bipartite graph in the space.

$$C_{nn} = P\{1, 2, \dots, n\} \times P\{1, 2, \dots, n\} \quad (7)$$

To analyze the scalability of such NIC interconnection models, particularly in systems where network expansion is anticipated, we consider the asymptotic behavior of inter-NIC connection growth.

$$L = \lim_{n \rightarrow \infty} \left| \frac{C_{nn+1}}{C_{nn}} \right| = \lim_{n \rightarrow \infty} \left| \frac{n^2+1}{n^2} \right| = 1 \quad (8)$$

This result implies that the marginal increase in interconnection complexity approaches a constant ratio of 1 as $n \rightarrow \infty$, which suggests a bounded growth rate of interconnection overhead.

Figure 4 presents a conceptual abstraction of an in-vehicle communication structure. In Figure 4a, a top-down view of the vehicle is shown, indicating the physical layout for control modules. Figure 4b models a logical communication hierarchy in which upper-layer

decision-making nodes interface with lower-layer wheel actuators through switches S_1 and S_2 .

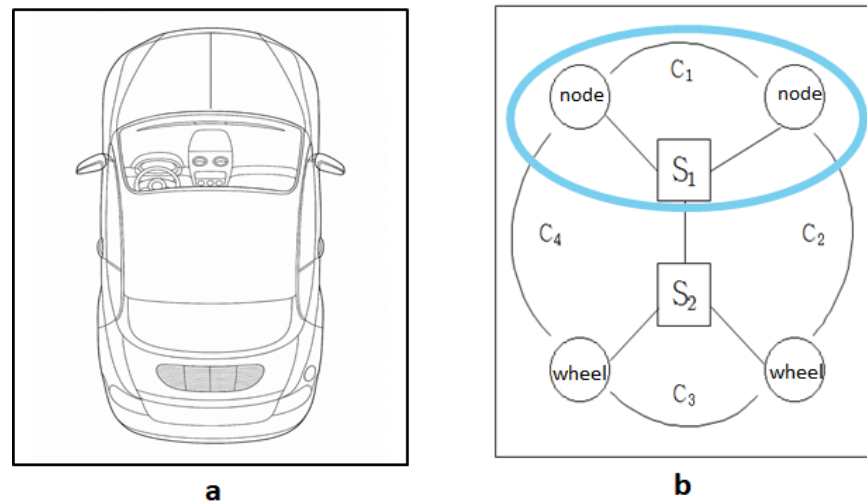


Figure 4. Conceptual diagram of a vehicle communication system. (a) Top-down view of a vehicle with embedded communication infrastructure. (b) Logical connection topology showing two node-level processors (top) connected to a central switch S_1 , and two wheel modules (bottom) connected via another switch S_2 . The connections C_1 to C_4 represent directed communication paths between components, where the upper nodes form the decision layer and the lower nodes correspond to actuator endpoints.

Let each connection between a module and switch be represented as follows.

$$C_i = (P_s, P_d), \text{ where } P_s \in \text{source node}, P_d \in S \quad (9)$$

The full set of communication paths in the system is defined as follows:

$$C = \{C_1, C_2, C_3, C_4\} \quad (10)$$

where C_1 and C_2 represent connections from decision nodes to switch S_1 . C_3 and C_4 represent connections from wheel actuators to switch S_2 .

To analyze in more detail, if we perform Hierarchical Layer Interpretation, it can be divided into the following three parts. (i) Upper Layer (Control Plane): Nodes interfacing through C_1 and C_2 to switch S_1 handle decision making, such as steering logic or velocity planning. (ii) Middle Layer (Switching Plane): Switches S_1 and S_2 form the backbone of communication, enabling data routing and isolation between layers. (iii) Lower Layer (Execution Plane): Actuators (wheels) receive control commands through C_3 and C_4 , enabling real-world action such as steering and traction.

In conclusion, designers of such communication systems can simulate communication links, evaluate bandwidth usage, and identify bottlenecks in the overall system by mathematically modeling the connection(C)-centric interactions across nodes (N) as well as specific ports P_i^a and P_j^b . This is especially important when designing multi-port NICs for area-wide automotive networks, multi-host RDMA clusters, or containerized microservice environments, where traffic patterns between nodes or ports are non-uniform and dynamically reconfigured. Such structures are also trending toward modeling real-world automotive Ethernet or ECU communication frameworks based on service-oriented architecture (SoA) [31,32].

4. Methodology

Related research on fault-tolerant high performance computing via a coding approach has been conducted by many researchers for a long time [33,34]. In addition to performing validation and verification at the entire system level, there are also scientists who use formal languages to perform formal validation at the chip level or gate level [35]. In such vehicular communication networks, it is also more important to efficiently check the stability of the system. In this paper, validation of the entire system of the vehicular communication network, i.e., INV, was considered as a priority. In order to verify the soundness of the system through the research results by fault injection [36], we focused on improving the algorithm [37] that selects an intended cyclic redundancy check (CRC) method to inject faults into the system and check.

Therefore, verification at the system level must be carried out, and the first priority for this is how to conduct the experiment. As shown in Figures 2 and 3, we must first decide on the initial number of NICs and the data to be transmitted accordingly. The experimental data were divided into integer-type data and character-type data for testing purpose, and the number of data used in the experiment was set to be the same. Therefore, in the actual experiment, five data sizes of 100, 343, 700, 5000, 10,000, 16,384, and 65,536 were applied to the system simultaneously. At this time, 100, 343, 700, 5000, and 10,000 are empirical values determined by the experiment, and 16,384 and 65,536 were determined by the pixel data size ratio compared to the 128×128 image, which is the image data with the same number of pixels, instead of avoiding the experiment by proportion. At this time, 128×128 means the number of image pixels.

As the first step, the port value of each NIC is initialized. The initialization function generates a value of between 1 and 4095 and artificially connects a random IP to the point where the error occurs [38].

Furthermore, we would like to explain the process of developing a fault diagnosis and recovery algorithm using cyclic redundancy check (CRC-16). In general, CRC-16 is a widely used error detection algorithm that processes a stream of input bits and computes a fixed-size 16-bit checksum (also called a codeword or remainder) using a predefined generator polynomial (in general, $x^{16} + x^{15} + x^2 + 1$). Algorithmic behavior should be explained as follows. CRC computation typically involves the following: first, the bitwise division of the input message polynomial by a fixed generator polynomial, and second, the use of XOR and shift operations at each step to update a register (often modeled as a Linear Feedback Shift Register, or LFSR). Also, depending on the implementation, two common variants exist: bitwise CRC, which processes each bit of the input sequentially, and bitwise or table-driven CRC, which processes each byte (or word) using precomputed tables, improving performance via lookup operations.

We present a simulation test that was used to conduct experiments on five types of errors using QPFT between NIC 1 and NIC 2. Among them, injection fault, data loss, injection noise, access fail, and wrong data are well expressed visually. The following is an algorithm simulation screen showing how the system detects and automatically recovers the errors that occurred.

Assuming that there are 5000 data in one file, we set it so that any piece of data, regardless of its type, causes a problem in the NIC. This is the stage where the fundamental fault-based validation approach that injects an intended fault begins. After that, the NIC sets an error at 0, and when initial loading is performed, the fault injection is rejected based on the research hypothesis that there are no data. At this time, we applied a method to check the health of the entire system through CRC-16, and the first data set that occurs overall may not be a problem. Each NIC is designed to be connected as shown in Figure 2. N_0 and N_1 are responsible for transmission to Port 2, and Port 3 is responsible for reception.

When receiving data from NIC 1, 2, and 3, the received data are 0, 1, and if the copies of data Port 0 and Port 1 are different, the CRC code of NIC_0 and NIC_1's Port 0 and Port 1 are compared to determine the conversion value of all data values by comparing the codes with the same port number before the NIC is judged normal. Also, as shown in Figure 3, the NIC is converted back to the end of the first NIC when designing the Ring-type topology of the current network structure, and the NIC information is transmitted.

The current program can always transmit packet data of CRC-18 bytes with CRC-16 or higher, and the padding length can perform the operation. The integer-type data and character-type data used in the experiment have the same integer value as the length of the data generated in the experiment, which is similar to the actual data length of all generated data. If the length of both data is different, the packet payload size may vary, so in the case of the widely used automotive CAN, Flex Ray, and UDP environments in the current network environment, the size of the 20-byte packet allows for the implementation of the padding function as the experiment progresses [39,40]. Also, time complexity analysis should be explained as follows. Let n be the total number of bits in the input packet. The complexity of CRC-16 computation can be analyzed in two common implementation approaches: bitwise and byte-wise (or table-driven). In the bitwise CRC-16 method, each bit of the input data undergoes a fixed sequence of operations: primarily shifts and XORs. Since each bit is processed individually in constant time, the total number of operations is proportional to the number of bits n , resulting in a time complexity of $O(n)$. In contrast, the byte-wise or table-driven CRC-16 implementation utilizes precomputed lookup tables to process data one byte at a time. Although only $n/8$ bytes are processed, each byte still requires a constant-time lookup and XOR operation. Therefore, the total operation count is $O(n/8)$ which simplifies to $O(n)$ in terms of asymptotic complexity. Thus, regardless of the implementation method, the CRC-16 algorithm exhibits linear time complexity relative to the input size, denoted as $O(n)$.

In the algorithm of the existing study [41], two ports were used when using a dual-port NIC [42]. However, in the new type of board, multiple ports (at least four ports) are used to build a communication environment between data. In addition to handling errors that occur on their own, we aim to implement a redundant switch to reduce the occurrence of errors. Partial or full redundant circuits are one of the causes of failures and can be problematic because they minimize the probability of stable operation [43].

The newly proposed quad-port NIC has the advantage of high transmission speed of at least two external ports. This overcomes the switching redundancy that can occur in the existing dual-port NIC itself and secures data transmission and system efficiency. Considering the importance of in-vehicle communication (INV) as well as the redundancy of external communication (OVN), the quad-port NIC design is essential for more stable data transmission. The currently implemented fault-tolerant program is structured in the form of a Mesh topology; refer to the proposed pseudo-code implemented in the existing way using the two ports at the top of the NIC system flow diagram shown in Figure 4 based on the program implementation using a quad-port, i.e., four-port NIC.

This pseudo-code shows the multi-port NIC communication process, including IP acquisition, data exchange per port, and integrity verification through CRC-16. When a mismatch is detected, an error notification containing the entire data block is broadcast, and all listeners respond via a predefined port, enabling a synchronized recovery or acknowledgement response. This mechanism supports reliable and fault-tolerant communication in embedded or distributed network systems.

The proposed pseudo-code (Algorithm 1) models a fault-tolerant communication scenario between two NICs, each equipped with four ports. Unlike previous dual-port approaches, the proposed quad-port design ensures fast deterministic timing and superior

fault tolerance. Communication is established between corresponding ports of NIC_i and NIC_j (i.e., $NIC_i.\text{port } k \leftrightarrow NIC_j.\text{port } k$), enabling parallel and isolated data exchange. A fault is intentionally injected into a specific port at a predefined iteration to evaluate the system's robustness. CRC16 (CCITT) is used to verify the integrity of transmitted and received data on each port, and if a mismatch is detected, a recovery mechanism is triggered to restore the corrupted data.

$$\begin{cases} C_i = M_i^* G_{crc} \\ C_i \neq C_{i-1} \\ C_i = C_{i-1} \end{cases} \quad (11)$$

For the interpretation related to CRC or linear fault detection code, consider the above Equations (1) to (3). Equation (1) represents the process of calculating the CRC value C_i of the current iteration by applying the generator polynomial G_{crc} to the given message M_i . Equations (2) and (3) are used as conditions for verifying the integrity of data by comparing the CRC values between consecutive iterations. That is, if $C_i \neq C_{i-1}$, there is a possibility that a data error has occurred, and if $C_i = C_{i-1}$, it means that the data have been maintained normally.

Algorithm 1. Fault-tolerant multi-port NIC communication process.

- Step 1. Initialize communication ports and assign IP addresses.*
 - Step 2. Establish port-to-port data exchange across multi-port NICs.*
 - Step 3. Perform CRC-16 verification for transmitted data blocks.*
 - Step 4. If mismatch detected, broadcast error notification.*
 - Step 5. Execute synchronized data recovery procedure using redundant or mirrored blocks.*
 - Step 6. Confirm restored CRC value and resume normal communication.*
-

The proposed algorithm enhances data recovery by simplifying CRC-based validation through the addition of a function that assigns and tracks a unique identifier for each NIC port [44]. This approach enables efficient data verification by associating CRC codes with specific ports. As shown in Equation (1) [45], CRC calculation is fundamentally a division operation. Each message block is encoded into an n -bit codeword by dividing a predefined bit string, represented as a polynomial, by a generator polynomial. The resulting remainder becomes the CRC value. This process is efficiently implemented using bitwise shift operations and modular polynomial arithmetic.

The algorithm employs the conventional CCITT CRC-16 library, which computes CRC values based on the assigned unique identifiers for each port [46]. By streamlining CRC computation and tracking, the method supports fast and accurate data integrity checks. In hardware implementations, such as those using Verilog, the CRC-16 algorithm is typically realized using a Linear Feedback Shift Register (LFSR) that updates in parallel with each clock cycle. This allows each byte of input data to be processed per clock cycle, enabling pipelined execution. While the overall latency of the CRC-16 operation remains $O(n)$ —where n is the number of input bits—the throughput can be significantly improved through pipelining or parallelization techniques. The CRC-16 computation logic, thus, maintains linear time complexity $O(n)$ with respect to the input packet size. This computational efficiency, combined with its minimal hardware resource requirements, makes CRC-16 particularly well suited for real-time and embedded systems, such as those found in automotive communication environments.

Moreover, the proposed algorithm addresses error detection and data loss by injecting and comparing data to identify corruption in stored information. Unlike previous approaches, which relied on retrieving and comparing stored CRC values after an error occurred [41], this method eliminates the inefficiencies of historical comparison. Each NIC

port and its associated memory perform real-time CRC checks, allowing for the immediate detection of transmission errors without incurring additional computational overhead.

5. Testing Environments

This section presents a network-based performance evaluation conducted within a CAN-based communication protocol test environment. The evaluation comprises three components: (i) the network configuration and experimental setup (Section 5.1), (ii) a response time analysis under varying frame rates to assess system scalability (Section 5.2), and (ii) a repeatability test under fixed frame conditions to evaluate stability and timing consistency (Section 5.3).

5.1. Experimental Setup

To emulate the architecture of an actual Controller Area Network (CAN)-based automotive system, the configuration depicted in Figure 5 was adopted.

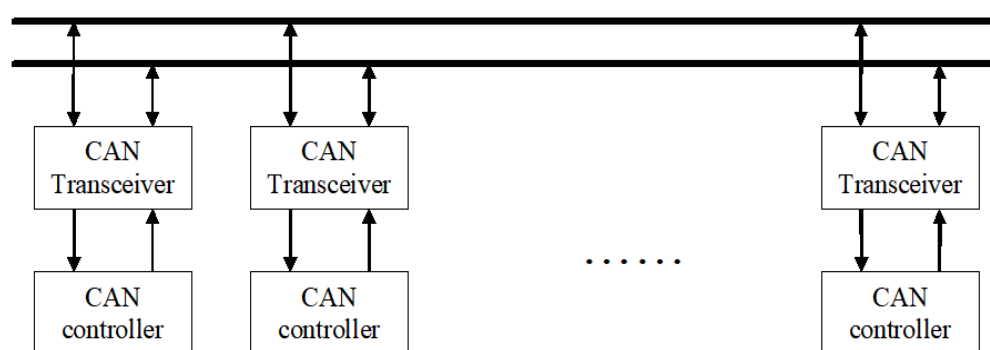


Figure 5. Network configuration for CAN-based communication among multiple nodes. Each node consists of a CAN controller interfaced with a CAN transceiver, all connected to a shared two-wire CAN bus, enabling distributed communication and message arbitration in accordance with the ISO 11898 standard.

To establish a baseline for understanding conventional in-vehicle networks (IVNs), Figure 5 illustrates a typical CAN bus architecture. In this setup, multiple CAN controllers communicate via transceivers over a shared two-wire differential line, implementing a broadcast-based message exchange mechanism. This architecture is widely employed in real-time control applications such as braking systems, powertrains, and body electronics. The experiments were conducted using the MPC5553MVB, a 32-bit high-performance microcontroller unit (MCU) developed by NXP. It integrates a PowerPC e200z6 core and supports modules such as eTPU, FlexCAN, and ADC, enabling the simulation of automotive control workloads. This MCU is specifically designed for real-time embedded applications that demand high reliability and compliance with automotive safety standards [47].

In this experiment, the real-time test module (MPC5553MVB) provides two CAN controllers (CAN_A and CAN_C), but only a single CAN transceiver is available. Consequently, all test scenarios were executed using the loopback function of the CAN controller. This approach allows for performance evaluations and response time testing within an isolated environment.

Figure 6 illustrates the internal loopback configuration of the CAN_A module. A unidirectional data flow is established between the transmitter and receiver over a virtual channel, indicated by the red line. This configuration facilitates intra-module communication, allowing for the validation of transmission integrity and timing behavior without relying on an external CAN bus. Such loopback testing is commonly used in hardware-in-

the-loop (HIL) simulations and fault injection scenarios to verify functional correctness in a controlled environment.

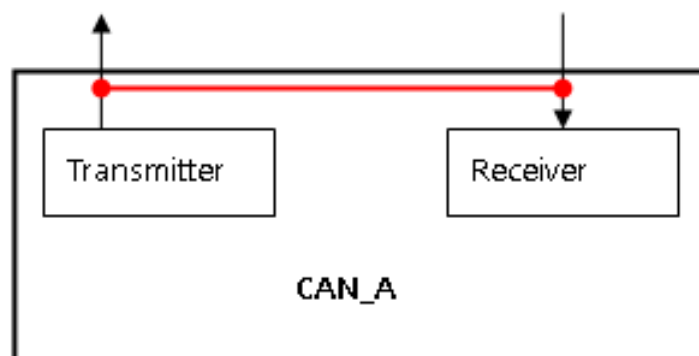


Figure 6. Internal communication structure of the CAN_A test node. The setup includes a dedicated transmitter and receiver communicating over a loopback CAN channel, allowing for isolated performance measurements and response time evaluations within a single CAN controller environment.

5.2. Response Time Test with Changing Frame Rate

A comprehensive performance evaluation was conducted using an MPC5553MVB-based embedded system to assess the system's scalability under increased transmission loads and its temporal response behavior. In this experiment, the transmitter continuously sends data frames of fixed length (8 bytes) to the receiver. At the receiver side, the total elapsed time is recorded from the arrival of the first frame to the arrival of the final frame. The results are expressed as ordered pairs (N) , where N denotes the total number of transmitted frames and t represents the corresponding cumulative transmission time.

Figure 7a illustrates the relationship between the number of transmitted frames and the corresponding transmission time (in milliseconds). As the frame count increases from 1000 to 9000, the elapsed time exhibits a near-linear trend, indicating that the system maintains consistent and predictable timing behavior under increasing data loads. This result confirms that the frame-handling logic of the MPC5553MVB-based communication system scales proportionally, without introducing significant timing jitter or latency. Figure 7b presents the average transmission speed (in bytes/s) measured over 16 repeated test cycles. Despite variations in frame count and repeated executions, the average throughput remains stable, consistently ranging between 5200 and 5500 bytes/s. This observation demonstrates the robustness of the system's internal buffering, scheduling, and timing mechanisms and validates its reliability for real-time communication tasks under controlled experimental conditions.

The primary objective of this experimental evaluation is to assess the responsiveness and timing determinism of the embedded system under variable transmission loads. By incrementally increasing the number of transmitted frames, this study examines the extent to which the system preserves real-time communication properties. A linear or near-linear growth in total transmission time t relative to the number of frames NNN confirms the system's ability to operate predictably. Conversely, deviations from this trend, such as nonlinear latency growth or elevated jitter, would suggest potential inefficiencies in the NIC architecture or limitations in the underlying in-vehicle network (IVN) protocol stack.

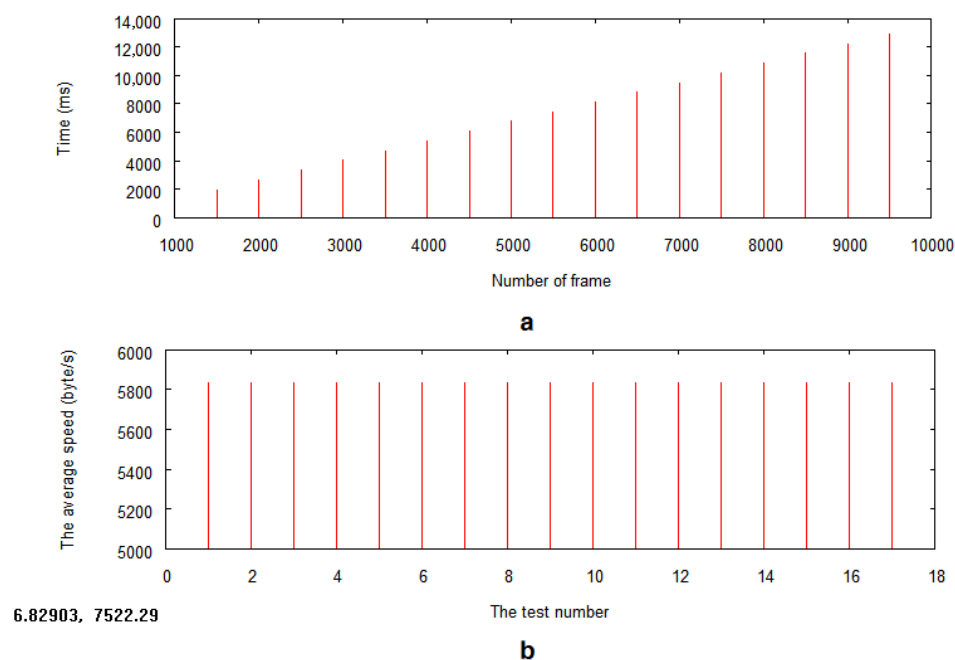


Figure 7. Transmission performance analysis of the MPC5553MVB-based test system. (a) Elapsed time in milliseconds versus the number of transmitted frames. (b) Average data transmission speed measured in bytes per second across multiple test iterations.

5.3. Repeatability Evaluation Test for Stability with Fixed Frame Configuration

This experiment is designed to assess the temporal stability and consistency of transmission performance under fixed data size conditions. Specifically, the transmitter repeatedly sends an image file with a constant size of 1010 bytes to the receiver. The procedures for data preparation, including frame segmentation and transmission scheduling, are detailed in the following section to ensure reproducibility and transparency in the evaluation process.

Figure 8 illustrates the two-stage process for generating a CRC-integrated header file used in embedded communication. In the first stage, a cyclic redundancy check (CRC) checksum is calculated for the entire image file using a dedicated CRC generation module. The resulting transmission payload includes both the original image data and the appended CRC value. In the second stage, these combined data are used to create a header file that defines the memory layout required by the target embedded system. All preprocessing steps are conducted on a host PC prior to deployment, ensuring efficient memory mapping and integrity verification during runtime.

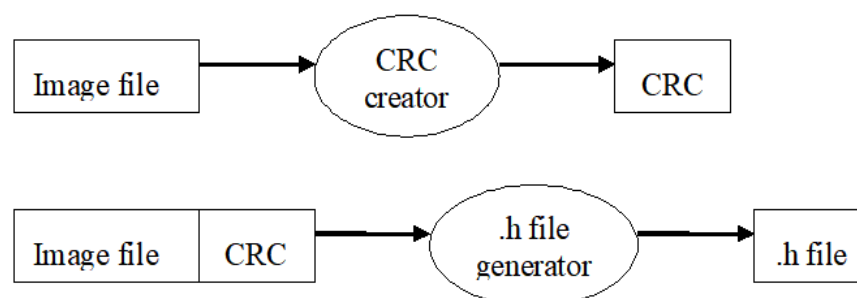


Figure 8. CRC-based header file generation process. The upper flow illustrates the generation of a CRC value from a given image file using a CRC creator. The lower flow shows how the image file and its associated CRC are combined to produce a corresponding C header file using a dedicated .h file generator tool.

To verify the correctness and integrity of data transmission between two nodes, a simple frame-based communication model was implemented between the transmitter and receiver modules. As illustrated in Figure 9, the transmitted data are encapsulated into a frame and delivered over the communication channel, where they are subsequently decoded and reconstructed at the receiver side.

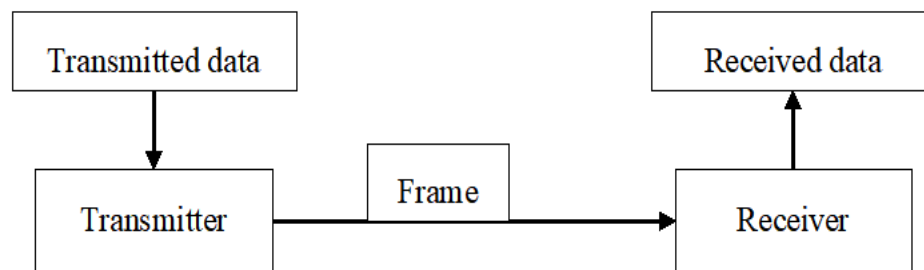


Figure 9. Conceptual model of frame-based data transmission for CRC validation. The transmitter packages the input data into a frame, which is transmitted to the receiver. The received data are then compared against the original transmitted data to verify communication integrity.

Following the generation of the header file that embeds both the image data and its associated CRC value, the subsequent phase involves verifying data integrity during transmission. As illustrated in Figure 10, the composite payload—comprising the image file and its CRC—is divided into fixed-length frames of 8 bytes, with the final frame potentially containing fewer bytes depending on the total data size. These frames are sequentially transmitted from the transmitter to the receiver, where they are reconstructed in order and temporarily buffered. Upon full reception, a CRC validation is executed to determine whether the reassembled data matches the original dataset, thereby confirming the integrity and reliability of the communication process. To evaluate the fault tolerance capabilities of the system, controlled error injection is applied during the transmission phase. Specifically, a deliberate bit-level error is introduced into one of the data frames to simulate a realistic communication fault. Upon receiving the altered payload, the system performs a CRC computation on the reconstructed data and compares it against the reference CRC embedded in the header. If a discrepancy is detected, the embedded error recovery logic is triggered, initiating a correction process using redundant or mirrored data blocks stored locally. This methodology facilitates not only real-time error detection, but also low-latency correction, thereby enhancing the overall robustness of the NIC-based communication architecture in embedded automotive systems. Accordingly, the total processing time—from the reception of the first data frame to the completion of CRC verification—is recorded and analyzed as a critical performance metric for ensuring communication reliability.

Figure 10 illustrates the outcomes of a repeatability test performed under fixed data load conditions on the MPC5553MVB-based embedded system. Figure 10a displays the transmission time, measured in milliseconds, over 20 consecutive test iterations. The transmission times exhibit minimal variance, consistently centering on 175 μ s, thereby demonstrating a high degree of temporal determinism. This stability reflects the predictability of the system's internal task scheduling and bus arbitration logic, with negligible timing jitter observed across repeated executions. Figure 10b shows the corresponding average transmission speed, measured in bytes per second, for each test iteration. The speed consistently falls within a narrow band between approximately 5300 and 5500 bps, demonstrating the system's ability to maintain throughput stability even under repetitive operation. The uniformity of both timing and speed metrics confirms that the communication and data handling routines in the firmware are robust against internal variations such as task switching, interrupt latency, or cache access overhead. These findings collectively

validate the robustness of the system's low-level transmission stack when operating under static frame sizes and fixed configuration parameters. Such characteristics confirm its suitability for real-time embedded control applications, where deterministic timing and consistent throughput are essential.

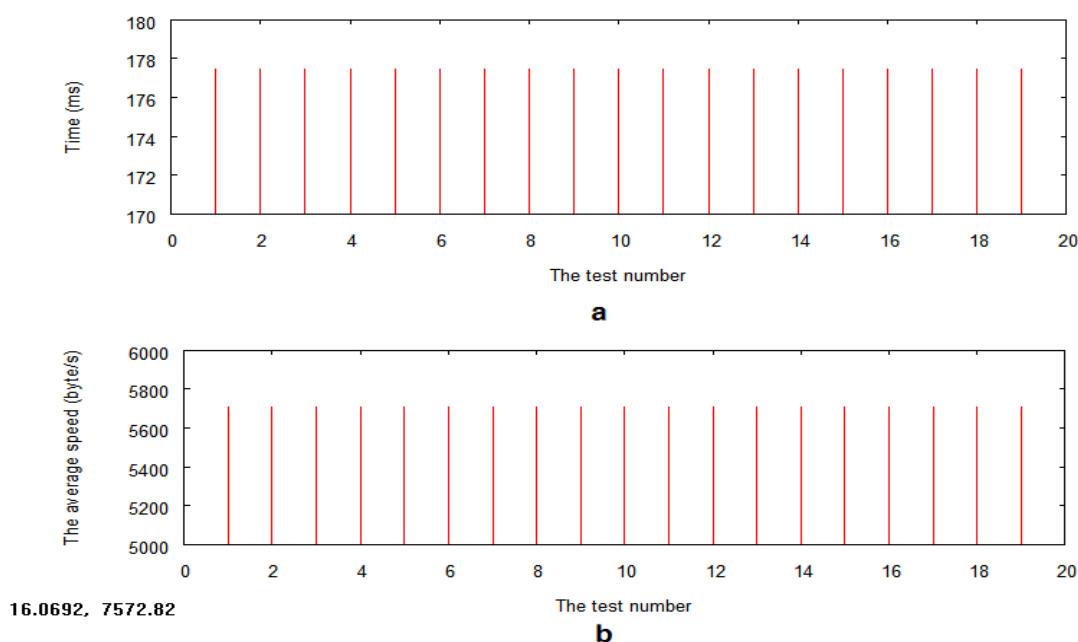


Figure 10. Stability analysis of transmission time and average speed under fixed frame conditions using the MPC5553MVB-based system. (a) Measured transmission time (μ s) across 20 repeated test iterations. (b) Corresponding average transmission speed (bytes/s) per test.

Furthermore, to provide a comprehensive evaluation of the MPC5553MVB-based embedded communication system, a two-wired BUS and bidirectional complementary test scenario compliant with ISO 11898 standards is designed, which is shown in Figure 11. Here, the number of transmission frames is gradually increased to investigate the scalability and fault tolerance of the system under various load conditions. In addition, the temporal stability and consistency of data throughput over consecutive transmission cycles are evaluated through repeatability analysis under fixed conditions.

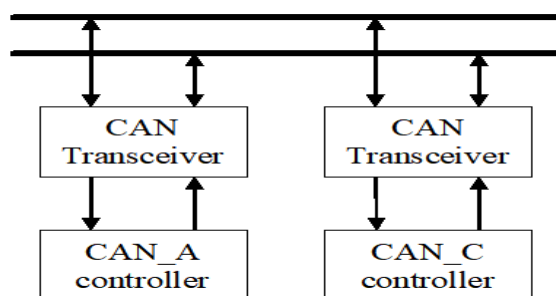


Figure 11. Experimental CAN network configuration using TJA1050 transceivers. The setup connects CAN_A and CAN_C controllers via high-speed CAN transceivers to a shared two-wire bus, enabling bidirectional communication and validating physical-layer interoperability in compliance with ISO 11898 standards.

Figure 11 shows a basic CAN (Controller Area Network) communication structure consisting of two CAN controllers (CAN_A and CAN_C), each connected to its respective CAN transceiver. The transceivers interface with a shared CAN bus, enabling bidirectional communication across all nodes. This topology demonstrates a typical multi-node CAN

configuration where messages are broadcast over the bus and selectively received by controllers based on message identifiers. Such an architecture is commonly used in automotive and industrial systems for robust and real-time data exchange.

In the frame scaling test (as shown in Figure 12a), the transmission time showed a nearly linear increase pattern as the number of frames increased from 1000 to 9000, which confirms that the data processing path of the system scales predictably with the input load. However, the average transmission rate remained consistent across all test instances, indicating that the communication interface and data bus maintain stable throughput characteristics even at higher frame rates. In contrast, Figure 12b shows very stable transmission behavior under static test conditions. The transmission time variation over 20 CRC repetitions was minimal and remained constant around 175 μ s. Likewise, the average rate was consistently in the range of 5300 to 5500 bytes/s. This high repeatability highlights the deterministic timing behavior of the system when external variables are controlled, suggesting strong internal synchronization and consistent peripheral performance. Comparing these two experiments, we can conclude that the MPC5553MVB-based embedded communications platform not only scales well with increasing data volumes, but also maintains robust performance during repetitive operations. The system exhibits both scalability and time stability, which are essential characteristics for real-time embedded applications such as automotive control systems that require both dynamic responsiveness and operational determinism. These results demonstrate its suitability for deployment in automotive INV and ONV environments where predictable latency and consistent throughput are essential. In addition, consistent performance across variable and constant workloads implies optimal configuration of the firmware design, memory access mechanisms, and bus arbitration strategies within the system. The upper plot, Figure 12a, compares the normalized transmission time between scalability testing (red, increasing frame count) and repeatability testing (blue, fixed-size iterative transmission). Scalability results show a linear increase in transmission time, indicating proportional load handling, while repeatability results maintain near-constant timing, highlighting temporal determinism.

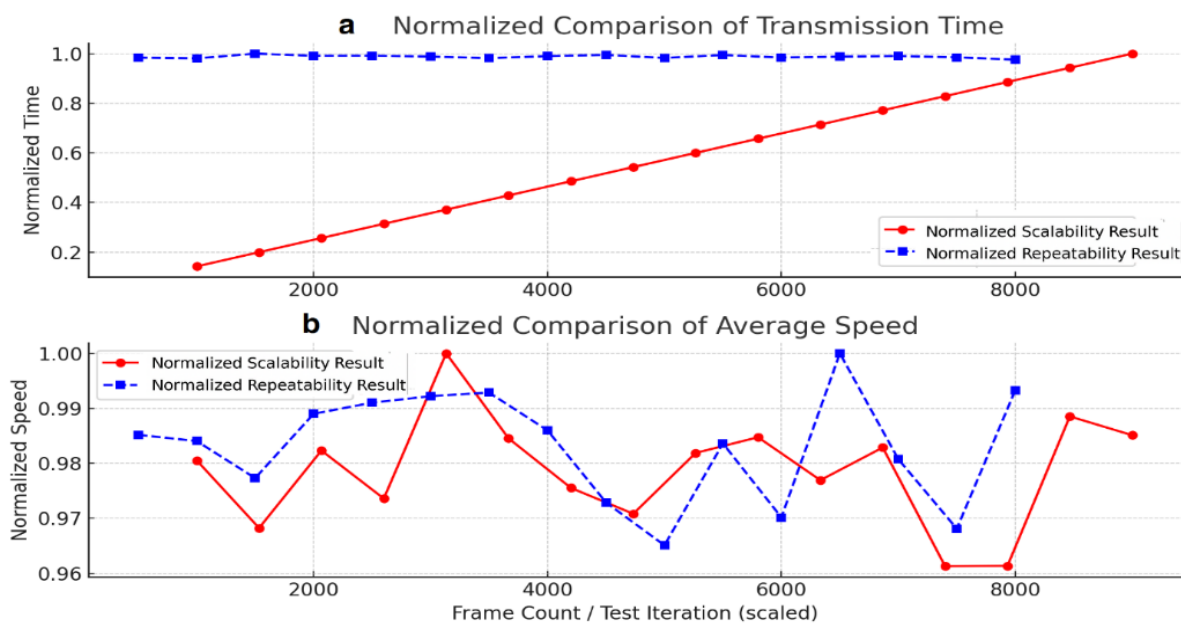


Figure 12. Normalized evaluation of transmission performance under scalability and repeatability conditions. (a) Normalized comparison of transmission time (b) Normalized comparison of average speed.

The lower plot, Figure 12b, illustrates normalized average speed across the same test conditions. Despite minor fluctuations, both scenarios demonstrate consistently high throughput. The repeatability condition yields slightly more stable speed performance, confirming the robustness of the NIC's transmission stack under static operational parameters. In conclusion, this is significant in that it visually shows the basic structure of a CAN network for stable and real-time data exchange between multiple nodes. CAN_A and CAN_C controllers communicate through a common bus, enabling efficient broadcast-based data transmission while maintaining independence between nodes, suggesting that this is a proven communication method in vehicle control systems or distributed control environments. The frame-scaling results demonstrate that the proposed quad-port NIC maintains a near-linear increase in transmission time as the number of frames increases from 1000 to 9000, with negligible timing jitter. Compared to conventional dual-port NIC architectures, which typically experience non-linear latency growth due to limited parallelism and increased bus contention, our design sustains deterministic timing even under higher loads. Furthermore, unlike BroadR-Reach (100BASE-T1) solutions optimized primarily for wiring cost reduction, the proposed architecture preserves throughput stability while supporting fault-tolerant operation.

In the repeatability tests under fixed-frame configurations, the proposed system consistently maintained transmission times tightly centered around 175 μ s and throughput between 5300 and 5500 bytes/s over 20 consecutive iterations. This performance not only exceeds that of commercial NICs such as RAPIEnet switches—which exhibit slightly higher jitter due to additional switching layers—but also confirms the proposed NIC's superior temporal determinism in embedded automotive environments. These results underline the system's advantage in real-time control scenarios, where predictable latency and high reliability are critical.

6. Quantitative Evaluation of Transmission Efficiency Based on Data Type Representation

We conducted the usability experiments of the proposed algorithm as follows. First, we controlled data transmission to verify the behavior of the integer and text data used. The data length used at this time may vary depending on the situation, and assuming that the code and control data continue to generate different data while the actual program is running, the length of the integer will be 0 to 255.

Similarly, assuming that the message data length and content processing speed vary over time, the *srand()* function is used to generate random data. Since the generated data is plain character data representing ASCII code values, the value is specified to be newly generated by increasing the lowercase letters when checking the readability of the data.

The content of the data is randomly generated, but the length of the payload data to be stored is optimized so that it can be implemented as a value of 0 in all CAN, FlexRay, and UDP environments. Thus, data are added to 18 bytes or 20 bytes of CRC-16 or higher to perform the operation [13].

The current program is substantially not to perform communication. When performing in the actual communication environment, it will take longer than this. At this time, the delay that occurs during the network connection process occurred about 3 μ s, and this was implemented by adding each numerical error to reduce the time taken to pass through the NIC switch [13]. In addition, when the experiment was conducted using three NICs, two test progress and transmission intervals occurred, resulting in a total delay time of 6 μ s. When the number of NIC switches was left, if a one-way transmission delay time occurred, the result was derived as in Equation (11).

To quantitatively represent the relationship between the number of NICs and the accumulated transmission delay, a delay propagation model is proposed. This model assumes a base delay of $3\ \mu\text{s}$ per NIC handover, excluding the final reception stage. Based on this assumption, the following expressions are derived to estimate the deterministic delay S_d and the final transmission time T_{fin} :

$$\begin{cases} S_d = 3\ \mu\text{s} \times \text{NIC}_{Num} - 1 \\ T_{fin} = S_d + R_{fin} \end{cases} \quad (12)$$

where S_d represents the cumulative deterministic delay introduced by intermediate NIC switches, calculated as $3\ \mu\text{s} \times \text{NIC}_{Num} - 1$, and T_{fin} denotes the final transmission time, obtained by summing S_d and the final reception timestamp T_{fin} .

This formulation enables accurate estimation of transmission latency in multi-hop Ethernet systems, especially under fault-tolerant or time-critical communication environments. As the number of NICs increases, the additive delay becomes a significant factor in real-time performance analysis and should be carefully considered in system design.

Furthermore, equation-based delay estimation offers a theoretical foundation for understanding system latency. To complement this, a packet transmission experiment was conducted to evaluate the actual data size variation according to data type.

The results, summarized in Table 2, compare integer-type and character-type file data under identical transmission conditions, providing insight into the overhead caused by data encoding formats.

The experiment compares integer-type and character-type data to evaluate how different data representations affect transmission efficiency under identical network conditions. The EPOCH value was fixed at 3000 to ensure consistent iteration cycles for fair comparison, minimizing variability due to temporal execution differences. This setting allows for an isolated analysis of how data type impacts packet volume and total transmission size across increasing packet counts.

Table 2. Packet transmission experiment (integer-type file data vs. character-type file data).

# of Packet Transmission	Data Size (Integer-Type File)	Data Size (Character-Type File)	EPOCH
100	457 bytes	501 bytes	3000
343	1567 bytes	1816 bytes	3000
700	3201 bytes	3386 bytes	3000
5000	22,580 bytes	25,225 bytes	3000
10,000	45,700 bytes	47,934 bytes	3000
16,384	74,908 bytes	81,689 bytes	3000

Figure 13 is a visual comparison of the change in data size (bytes) according to the increase in the number of packets for integer-type data and character-type data, respectively. It can be seen that character-type data require a larger data size than integer-type data throughout the entire section, which reflects the characteristic that character data consume more bytes for expression than integer data. In particular, this difference becomes more prominent as the number of packets increases, and it suggests that the selection of data types can have a substantial impact on communication efficiency, given that character-type data have a higher transmission cost in the same EPOCH environment, 3000.

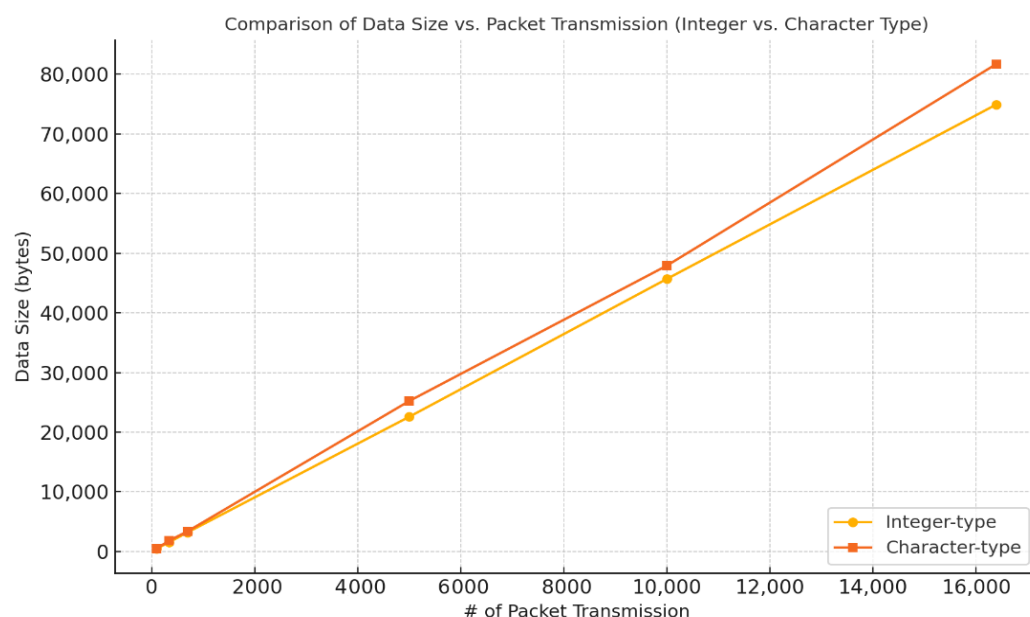


Figure 13. Comparison of data size versus the number of packet transmissions for integer-type and character-type data. Character-type data exhibit a consistently larger size overhead due to additional encoding, and both axes include thousand-separator formatting for improved clarity.

The evaluation of packet transmission across varying data sizes serves to enhance the credibility of hardware-level performance assessment. Given the experimental reliance on FPGA-based simulation and loopback trials, it is critical to clarify the inherent limitations of loopback configurations. By emphasizing the controlled nature of the automotive testbed conditions, this methodology strengthens the validity of the experimental results and reinforces the trustworthiness of the proposed architecture under realistic operational constraints. In this study, the loopback configuration was implemented with a fixed buffer size of 1024 bytes and a transmission–reception cycle period of 5 μ s per frame. The test environment utilized a deterministic timing window with jitter tolerance below ± 0.3 μ s to ensure repeatable signal propagation. A total of 10,000 loopback iterations were conducted, with each cycle recording latency, frame integrity, and CRC verification metrics. This quantifiable setup provides a reliable basis for benchmarking transmission stability and fault detection precision in real-time automotive communication systems.

7. Hardware Implementation and Validation of CRC-Corrected Ethernet Architecture

The increasing demand for high-reliability and high-bandwidth communication in modern vehicles has necessitated the development of fault-tolerant NIC architectures suitable for IVN environments. This chapter presents a detailed implementation and verification of a real NIC system designed to operate under deterministic conditions, providing resilient communication in automotive Ethernet applications.

Achieving real-time determinism in distributed NIC architectures mandates precise clock synchronization across nodes. The IEEE 1588 precision time protocol (PTP) is employed to enable ≤ 1 μ s threshold synchronization [11,27,28]. To implement this, a complete Ethernet MAC layer along with a software communication stack must be realized. The Synopsys Ethernet MAC IP core, which supports FIFO buffering, CRC validation, and traffic prioritization, is integrated and validated using RTL simulation tools [26,30].

To ensure uninterrupted communication under fault conditions, the NIC architecture incorporates the Redundant Ring Protocol (RRP) in compliance with IEC 62439-7. RRP provides seamless path redundancy and immediate failover in Ring topologies, which is

essential for critical subsystems in IVNs such as gateway ECUs and sensor clusters. Studies have demonstrated that RRP achieves near-zero recovery time, reinforcing its suitability for real-time automotive networks [29].

Also, the NIC implementation utilizes the Xilinx Zynq-7000 extensible processing platform (EPP), which combines a dual-core ARM Cortex-A9 processor running at 900 MHz with reconfigurable logic fabric [30,47]. The programmable logic is leveraged to implement custom MAC control modules, CRC generation units, and high-speed recovery logic. AXI interconnections ensure efficient communication between processing and logic domains.

Commercial-grade IVN NIC devices, such as RAPIEnet-based Ethernet switches, demonstrate the practicality of the proposed architecture. Typical configurations include 1 Gbps Ethernet with four-port switching capabilities, integrated TCP/IP recognition, CRC validation, and PWM signal handling [30]. The underlying device driver stack utilizes Synopsys MAC IP cores with tri-level FIFO queues for differentiated quality of service [31]. While commercial-grade IVN NICs such as RAPIEnet-based Ethernet switches offer integrated support for CRC validation, tri-level FIFO queuing, TCP/IP protocol recognition, and real-time PWM signal processing, their design is often optimized for deterministic in-vehicle control with minimal latency and fast error recovery within a 1 μ s threshold. In contrast, TSN (Time-Sensitive Networking) architectures—based on IEEE 802.1 standards—provide time-aware scheduling and bounded latency guarantees over standard Ethernet but typically rely on centralized time synchronization and software-based traffic shaping, which can introduce added complexity in distributed systems. Furthermore, TTEthernet (Time-Triggered Ethernet), widely adopted in aerospace and high-integrity automotive systems, offers strict time-slot-based scheduling and fault-tolerant redundancy at the protocol level [33]. However, TTEthernet's deterministic behavior is achieved through a rigid time-triggered architecture, which can limit scalability and flexibility compared to RAPIEnet's port-level redundancy and modular integration. Therefore, while TSN and TTEthernet emphasize synchronization and timing determinism at the network level, RAPIEnet distinguishes itself through hardware-centric fault detection and correction, enabling practical deployment in harsh automotive environments with high resilience and minimal computational overhead. These distinctions suggest that the choice of NIC architecture should be closely aligned with the intended application domain, favoring RAPIEnet for multi-port low-latency automotive use cases, and TSN or TTEthernet for systems requiring global temporal determinism or mixed-criticality scheduling.

Moreover, we also compare RAPIEnet and BroadR-Reach. Both are used in automotive Ethernet applications [33], but they serve different system-level goals: First, RAPIEnet is ideal for deterministic fault-tolerant systems, such as safety-critical subsystems, thanks to its real-time Ring topology and redundancy support. Second, BroadR-Reach (IEEE 802.3bw) is better suited for non-critical high-volume data applications like infotainment, camera-based ADAS, and sensor streaming, where cost and weight efficiency are more important than hard real-time guarantees. In summary, RAPIEnet and BroadR-Reach represent two distinct approaches to in-vehicle Ethernet communication. RAPIEnet offers robust fault-tolerant capabilities with deterministic performance, making it ideal for safety-critical and real-time control applications. In contrast, BroadR-Reach provides a cost-effective lightweight solution optimized for bandwidth-efficient tasks such as infotainment and ADAS. Selecting the appropriate protocol depends on the system's real-time requirements, fault recovery needs, and architectural constraints within the automotive network.

The NIC system incorporates a fault detection and correction mechanism composed of three key modules: a data comparison unit for validating cyclic redundancy check (CRC) values across multiple ports, a memory unit for temporarily storing verified data frames, and a response generator designed to issue corrective signals within a response time of

1 μ s, as we presented in Section 5. Simulation results confirm that the system can detect and recover from data corruption within the 1 μ s threshold. These outcomes are validated through waveform analysis using Mentor Graphics ModelSim [48,49].

Therefore, this chapter demonstrates the design, implementation, and validation of a fault-tolerant NIC architecture suitable for automotive Ethernet environments. The integration of IEEE 1588 synchronization, RRP failover mechanisms, and CRC-based data correction provides a robust foundation for future IVN deployments. The Zynq-7000 EPP platform offers the necessary processing and reconfigurable resources for real-time diagnostics and error recovery.

To further evaluate the temporal stability and quality of service (QoS) under different network configurations, jitter metrics were also analyzed and are summarized in Table S4 (in Supplementary Materials). This complements the end-to-end delay analysis in Table S3 by providing insight into the variance in packet arrival times, which is critical for assessing real-time communication performance in automotive Ethernet environments. Following the end-to-end delay assessment in Table S4, Table S5 provides a complementary analysis of jitter characteristics, which is essential for understanding the consistency of data transmission under the Double Star and Ring topologies.

Table S3 and Table S4 present the experimental results of this study, comparing end-to-end delay and time jitter across Double Star and Ring network topologies to evaluate overall communication performance. While the Ring topology showed slightly better performance for VoIP in terms of both delay and jitter, the Double Star configuration consistently outperformed it in nearly all other application types, particularly in real-time and control-critical transmissions. The key distinction lies in stability and predictability: the Double Star topology provided lower latency and more consistent delivery, making it more suitable for time-sensitive automotive applications.

To further demonstrate the effectiveness of the proposed NIC's fault detection and correction mechanism under real-time constraints, Figure 14 illustrates the internal state transitions during error injection and subsequent recovery. This simulation snapshot highlights how corrupted data are identified through CRC mismatch and subsequently repaired within one iteration, verifying the system's ability to autonomously restore data integrity in ≤ 1 μ s threshold timescales.

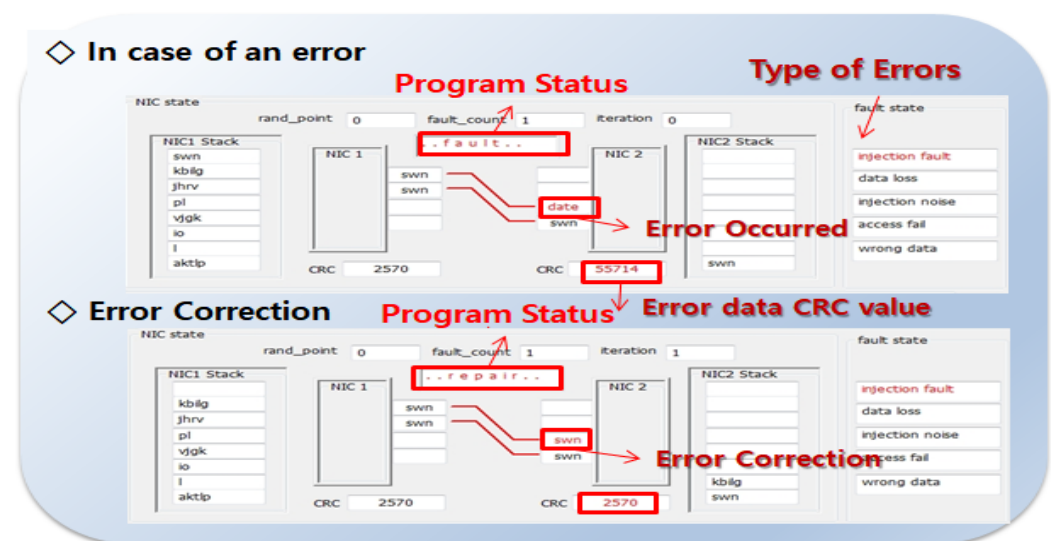


Figure 14. Illustration of error detection and correction process in the NIC communication system. The upper panel shows the system detecting a transmission error with mismatched CRC and fault status flags. The lower panel demonstrates the successful recovery of the corrupted data and restoration of the CRC value through error correction mechanisms.

In detail, Figure 14 illustrates the fault-tolerant behavior of the NIC communication system during error occurrence and recovery. In the top section, an error is detected during data transmission, indicated by the program status “.fault” and a mismatched CRC value, i.e., 55714, corresponding to an “injection fault” in the fault state. In the bottom section, the system enters the “.repair” state, and the corrupted data are restored to its correct form, verified by the restored CRC value, i.e., 2570. This confirms the system’s ability to detect transmission errors and autonomously perform data correction to maintain communication integrity.

The waveform in Figure 15 presents a digital timing simulation generated using Mentor Graphics’ ModelSim [48,49], illustrating the behavior of an NIC architecture during data transmission. The CRC outputs of both NIC 1 and NIC 2 are shown to remain in a high-impedance (‘Z’) state, indicating that the CRC computation module was not yet functionally implemented at the time of simulation. In contrast, the bottom section of the waveform confirms the successful reading of external file data into the system, which is subsequently used as dynamic input to the data path logic. This simulation validates the integrity of file-based data injection while concurrently highlighting a critical missing functional component—CRC logic—for complete transmission verification.

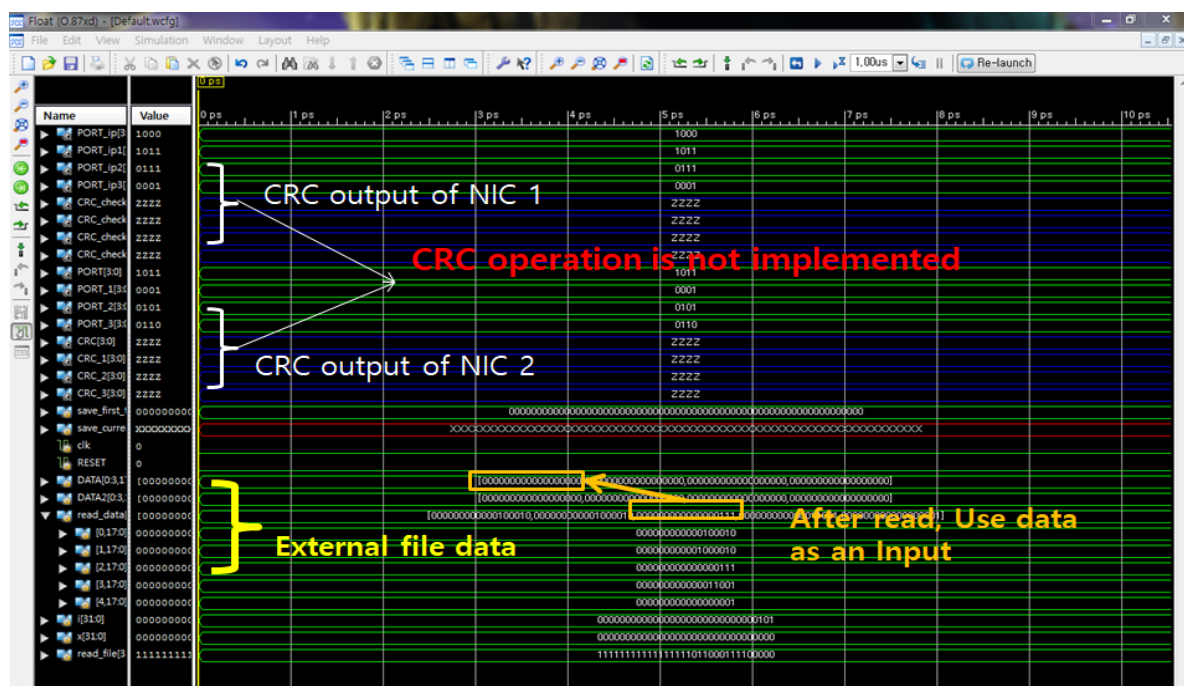


Figure 15. Simulation waveform, captured from ModelSim, showing the CRC outputs of two NICs and the external file data used as input. The CRC fields remain constant (indicated as ‘Z’) due to unimplemented CRC logic, while the read operation successfully loads external file content for subsequent use in the transmission pipeline.

Figure 16 illustrates a Verilog-based test bench structure and data injection logic designed for evaluating fault detection and recovery mechanisms in Network Interface Controllers (NICs). The simulation reads hex-formatted input and controls test progression based on a counter mechanism triggered on each positive clock edge. In Figure 16a, the hierarchy confirms the instantiation of two NIC modules, each containing four distinct communication ports, reflecting a parallelized communication environment. Figure 16b shows the associated behavioral Verilog code, where external input data are sequentially read from a file and injected into the NIC under test on every rising clock edge. This setup enables systematic fault simulation and recovery validation by automating input

vector delivery and integrating a controlled test progression counter, thereby facilitating reproducible hardware validation for real-time data path integrity.

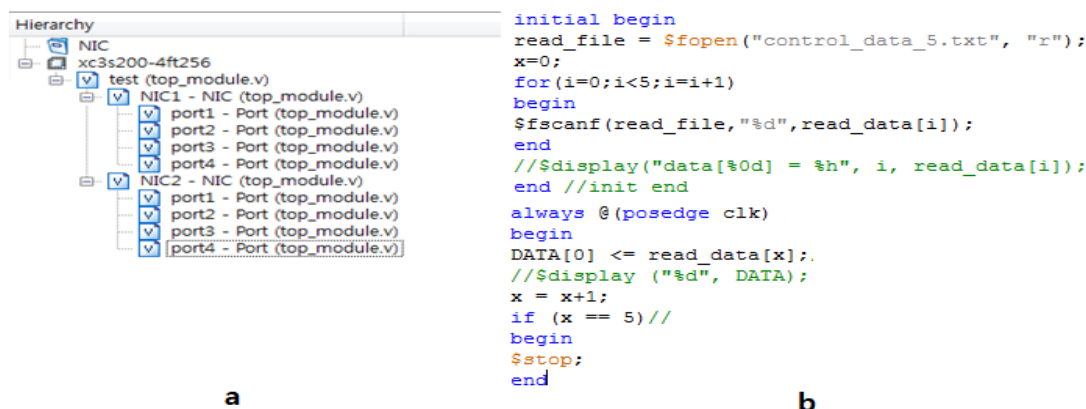


Figure 16. Verilog-based implementation for data-driven fault injection testing. (a) Module hierarchy showing two NIC instances, each with four ports. (b) Partial Verilog code snippet that initializes input from an external file and feeds it into the NIC module for sequential fault detection and repair evaluation.

The environment for testing the fault tolerance program in the above manner is the one that verifies the algorithm to be applied in an NIC environment with four ports without applying socket communication. For the actual simulation, the method was changed to use the Verilog codes in Figure 13. In addition, the algorithm that prioritizes and processes the transmission of various types of data, such as integer-type or non-integer type file data, as well as the control and message data currently being handled, was verified by testing with actual data.

Figure 17 illustrates the functional simulation results of a 16-bit CRC computation module, as observed in a waveform generated using the commercial tool Mentor Graphics ModelSim [25]. The signal `data_in [7:0]` represents sequential byte-level input data applied synchronously with the rising edge of the system clock (`clk`), while the `crc_en` signal enables CRC calculation. The evolving `crc_out [15:0]` values show valid transitions corresponding to the applied input, confirming the correct behavior of the LFSR-based CRC generator. The waveform also indicates that reset (`rst`) is held inactive, ensuring uninterrupted register operation, which validates the module's ability to perform real-time CRC encoding in streaming data environments.

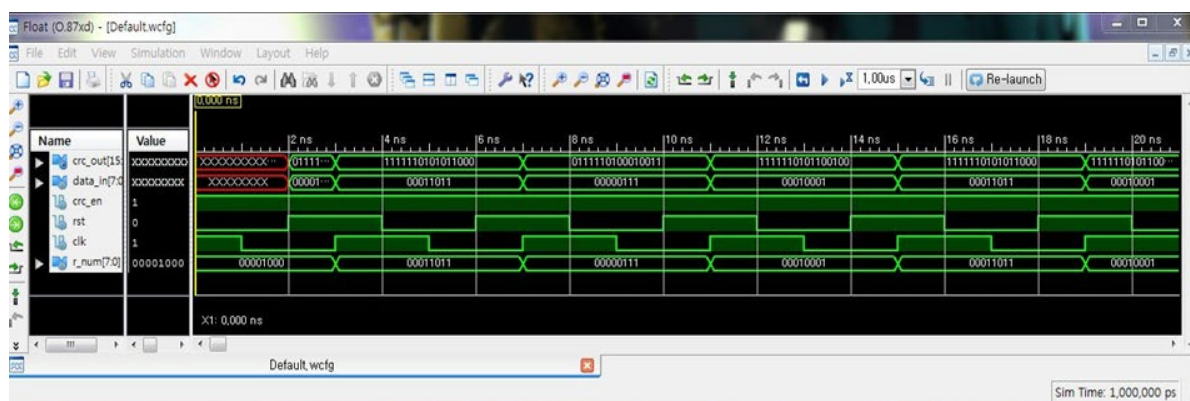


Figure 17. RC module waveform simulation generated using Mentor Graphics ModelSim. The waveform shows the dynamic behavior of `crc_out [15:0]` in response to eight-bit input `data_in [7:0]` under active clock (`clk`) and control signals (`crc_en`, `rst`). The correct transition of CRC outputs demonstrates functional correctness of the implemented LFSR-based CRC computation logic.

Furthermore, Figure 18 provides a detailed waveform-level view of an NIC-based fault injection simulation, capturing multiple signal transitions during port communication and CRC monitoring. The PORT_ip and PORT_2 signals exhibit synchronized transitions driven by external data injection, confirming the successful sequential loading of test vectors from a memory-mapped file. Notably, the CRC_check lines remain in a high-impedance state ('Z'), indicating that the CRC generation or comparison logic is either disabled or not yet implemented at this simulation stage. The temporal alignment of the clk, RESET, and read_data signals ensures reproducible timing control, allowing for the precise evaluation of functional behavior under controlled fault scenarios within a digital hardware test bench environment. The synchronization of data loading and clock edges demonstrates deterministic timing behavior for test-bench-controlled input delivery.

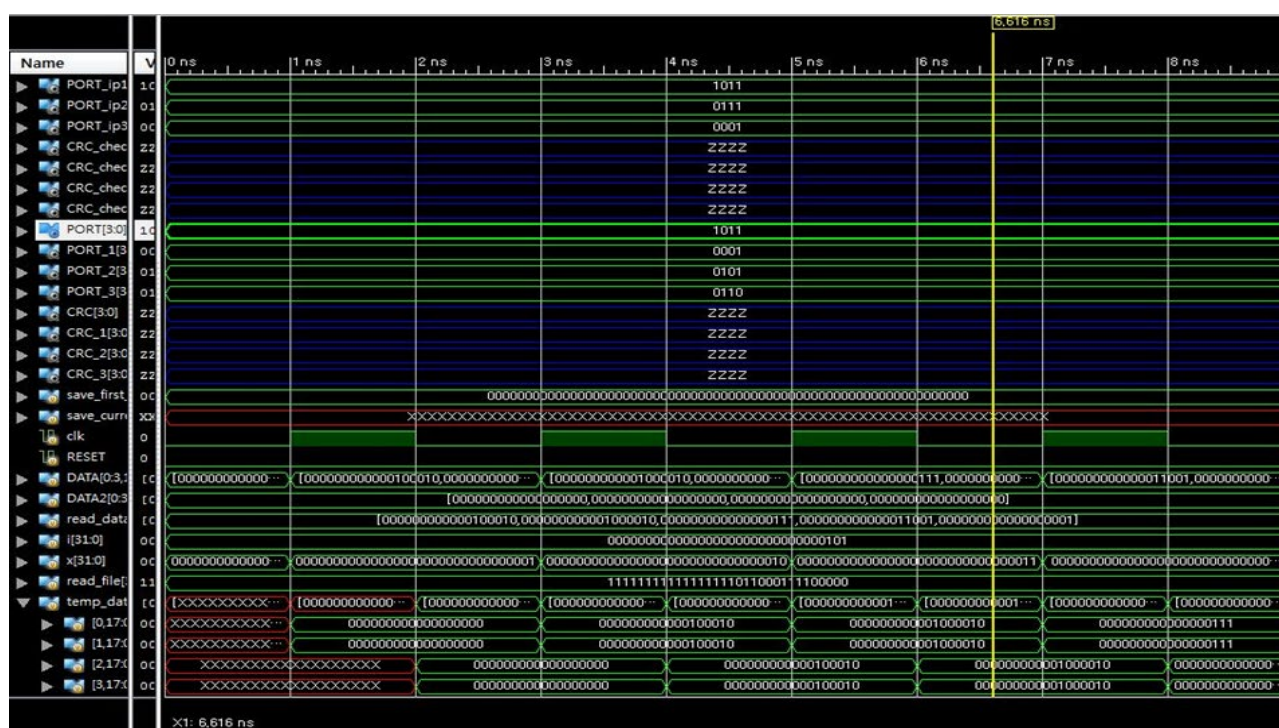


Figure 18. Simulation waveform, captured from ModelSim, showing port-level data transfer, CRC verification channels, and memory-mapped file input for NIC operation, captured using Mentor Graphics ModelSim. The signals PORT_ip and PORT_2 illustrate input data propagation, while CRC_check lines remain in a high-impedance ('Z') state, indicating pending CRC computation. Simultaneously, external file data are read and mapped to internal variables (DATA, read_data, temp_data) for fault injection and verification.

Therefore, the experimental results and waveform analyses collectively validate the functional correctness and timing determinism of the proposed fault-tolerant NIC system architecture. Through file-driven input vector injection, LFSR-based CRC logic, and real-time recovery mechanisms, the system demonstrates robustness in detecting and repairing corrupted data across multi-port communication paths. The high-impedance CRC outputs observed during simulation emphasize the need for complete CRC pipeline integration, while the synchronous behavior across clock domains ensures reliable data propagation. Overall, the hardware–software co-simulation framework effectively supports controlled fault evaluation and lays the groundwork for scalable real-time embedded network diagnostics.

8. Conclusions

In this study, we implemented and verified a fault-tolerant multi-port NIC architecture designed specifically for automotive applications. Building upon the limitations of conventional dual-port NICs, our system introduces a four-port or higher NIC configuration integrated with a specialized fault tolerance algorithm for in-vehicle networks (IVNs). This architecture was tested within a dedicated verification framework using a combination of control data, message data, and sound source data to assess its real-time operational reliability.

Experimental results demonstrated effective CRC-based error detection and recovery mechanisms. Notably, control data exhibited slightly faster processing times compared to message data. However, the efficient handling of message data in low-bandwidth environments suggests that the system can maintain high responsiveness where lightweight transmission is critical. The framework was verified through RTL-level simulation using Verilog HDL, file-driven data injection, and waveform analysis, confirming both functional timing accuracy and system scalability. Compared to conventional dual-port NICs, the proposed design offers increased parallelism, reduced switching delays, and superior scalability in high-load environments. Unlike BroadR-Reach (100BASE-T1), which prioritizes wiring cost reduction over fault tolerance, our NIC maintains real-time determinism while supporting CRC-based error detection and sub-microsecond recovery. In contrast to industrial-grade RAPIEnet solutions, the proposed architecture achieves comparable or better temporal stability with lower jitter while remaining optimized for automotive control workloads.

This work contributes a robust solution for fault-tolerant communication in automotive Ethernet systems operating at 1 Gbps or higher. While initial simulations were conducted under simplified conditions, the implementation lays the foundation for real-world deployment in next-generation automotive platforms. These contributions collectively position the proposed NIC as a high-performance fault-resilient solution for next-generation IVNs requiring both scalability and deterministic operation.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/act14080391/s1>, Figure S1: Methodology used in fault tolerance system (a) Data transmission method (b) Data restoration method after finding error, Figure S2: Verilog implementation of a 16-bit CRC generator module based on the polynomial $x^{16} + x^{15} + x^2 + 1$. The design uses a parallel Linear Feedback Shift Register (LFSR) architecture and XOR logic to compute CRC values over 8-bit input data, allowing real-time error detection during transmission; Table S1: Summary of Connection-oriented Model and its description, Table S2: Experimental Environment for Fault Diagnosis and Recovery Algorithm, Table S3: End-to-end delay values for packets of all applications in Double Star and Ring Topologies (in milliseconds), Table S4: Time jitter values for packets of all applications in Double Star and Ring topologies (in milliseconds), Table S5: Comparison Result RAPIEnet and BroadR-Reach (in terms of Data Rate and Physical Layer Characteristics).

Author Contributions: S.L. was responsible for data analysis, development of stimuli, and drafting of the model. S.Y. interpreted the results, and S.Y. and T.J. supervised the project and drafted the manuscript (review and editing). The authors would like to thank Eunji Jang, Bui. Hui, and Seunghyn Baek, who were full-time graduate students in the Advanced Intelligent Research (AIR) laboratory, for providing detailed drawings and simulations. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Hallym University Research Fund, 2025(HRF-202506-005).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available on request due to restrictions, e.g., privacy or ethical.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Nadeem, T.; Dashtinezhad, S.; Liao, C.; Iftode, L. TrafficView: Traffic data dissemination using car-to-car communication. *ACM SIGMOBILE Mob. Comput. Commun. Rev.* **2004**, *8*, 6–19. [\[CrossRef\]](#)
2. Zeng, W.; Khalid, M.A.; Chowdhury, S. In-vehicle networks outlook: Achievements and challenges. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1552–1571. [\[CrossRef\]](#)
3. Martínez-Fernández, S.; Ayala, C.P.; Franch, X.; Nakagawa, E.Y. A Survey on the Benefits and Drawbacks of AUTOSAR. In *Proceedings of the First International Workshop on Automotive Software Architecture*, Montreal, QC, Canada, 4–8 May 2015; pp. 19–26.
4. Stepanovic, M.; Bjelica, M.; Kastelan, I.; Velikic, G. AUTOSAR: Foundation and development of automotive standardized platform. In *Proceedings of the 2020 IEEE International Conference on Consumer Electronics (ICCE)*, Phu Quoc Island, Vietnam, 13–15 January 2020; IEEE: Piscataway, NJ, USA, 2020.
5. Ferrant, J.L.; Gilson, M.; Jobert, S.; Mayer, M.; Montini, L.; Ouellette, M.; Rodrigues, S.; Ruffini, S. *Synchronous Ethernet and IEEE 1588 in Telecoms: Next Generation Synchronization Networks*; John Wiley & Sons: Hoboken, NJ, USA, 2013.
6. Peserico, G.; Morato, A.; Tamarin, F.; Vitturi, S. Functional safety networks and protocols in the Industrial Internet of Things era. *Sensors* **2021**, *21*, 6073. [\[CrossRef\]](#)
7. Abbas, S.H.; Hong, S.H. A top-down approach to add hot-pluggable asynchronous devices to RAPIEnet infrastructure. In *Proceedings of the 2009 9th International Symposium on Communications and Information Technology*, Incheon, Republic of Korea, 28–30 September 2009; pp. 128–133.
8. Olsson, B.; Andersson, J. Next-generation automotive networking: Bandwidth requirements and performance evaluation. *IEEE Trans. Veh. Technol.* **2023**, *72*, 3023–3035.
9. Kasap, D.; Carpegna, A.; Savino, A.; Di Carlo, S. Microarchitectural markers for radiation-induced soft-error detection in embedded real-time systems. *IEEE Trans. Comput.* **2022**, *71*, 1289–1302.
10. Bello, L.L. The case for ethernet in automotive communications. *ACM SIGBED Rev.* **2011**, *8*, 7–15. [\[CrossRef\]](#)
11. Lee, H.; Jabbar, H.; Lee, S.; Choi, S.; Lie, Q.; Kim, I.; Choi, S.; Park, D.; Min, S.; Lee, Y.; et al. Ethernet-based Communication Architecture Design and Fault-Tolerant System. In *Proceedings of the International Multi Conference of Engineers and Computer Scientists*, Hong Kong, 14–16 March 2012; Volume 2, pp. 1181–1184.
12. Kim, I.; Jeong, T. Educational Use Cases and Standard Analysis of Smart Car Applications. In *Proceedings of the Korean Computer Education Society Winter Conference*, Seoul, Republic of Korea, 2 September 2013.
13. Jang, E.; Kim, I.; Lee, W. Synchronize Ethernet-based Fault Injection Algorithm Implementation for Intelligent Automotive Network. *J. Internet Comput. Serv. J. Korea Internet Inf. Soc.* **2016**, *17*, 43–50.
14. IEEE 802.3ch Multi-Gig Automotive Ethernet PHY Task Force. IEEE P802.3ch: Multi-Gig Automotive Ethernet PHY Update Report (Version 1.2). IEEE. March 2020. Available online: <http://ieee802.org/3/ch> (accessed on 2 August 2025).
15. Available online: <https://www.broadcom.com/products/ethernet-connectivity/automotive/phy/bcm89810> (accessed on 2 August 2025).
16. Bagbaba, A.C.; Jenihhin, M.; Raik, J.; Sauer, C. Efficient fault injection based on dynamic HDL slicing technique. *arXiv* **2020**, arXiv:2002.00787.
17. Mihalič, F.; Truntič, M.; Hren, A. Hardware-in-the-Loop Simulations: A Historical Overview of Engineering Challenges. *Electronics* **2022**, *11*, 2462. [\[CrossRef\]](#)
18. Kanawati, G.A.; Kanawati, N.A.; Abraham, J.A. FERRARI: A Flexible Software-Based Fault and Error Injection System. *IEEE Trans. Comput.* **1995**, *44*, 248–260. [\[CrossRef\]](#)
19. Wang, L.-T.; Stroud, C.E.; Toubia, N.A. *System-on-Chip Test Architectures: Nanometer Design for Testability (Systems on Silicon)*; Chapter 3, Fault-Tolerant Design; Morgan Kaufmann: Burlington, MA, USA, 2010.
20. Combes, P.; Caron, P. Fuzzy Petri Nets for Traffic Node Reliability in Autonomous Vehicles. *Sensors* **2024**, *24*, 6337. [\[CrossRef\]](#)
21. Marani, R.; Cantoni, M.; Tiloca, M. Software–Hardware Embedded System Reliability Modeling with Length and Failure Dependency. *Microprocess. Microsyst.* **2023**, *101*, 104123.
22. Solouki, M.A.; Angizi, S.; Violante, M. Dependability in Embedded Systems: A Survey of Fault-Tolerance Methods and Software-Based Mitigation Techniques. *J. Real-Time Syst.* **2024**, *7*, 11–35. [\[CrossRef\]](#)
23. Kriebel, S.; Markthaler, M.; Granrath, C.; Rumpe, B. Modeling hardware and software integration by an advanced digital twin for cyber–physical systems: Applied to the automotive domain. In *Handbook of Model-Based Systems Engineering*; Madni, A., Augustine, N., Sievers, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2023; pp. 1–38. [\[CrossRef\]](#)

24. Danielis, P.; Parzyjegl, H.; Mühl, G.; Schweissguth, E.; Timmermann, D. Frame replication and elimination for reliability in time-sensitive networks. *arXiv* **2021**, arXiv:2109.13677.
25. Lo Bello, L.; Patti, G.; Leonardi, L. A perspective on Ethernet in automotive communications—Current status and future trends. *Appl. Sci.* **2023**, *13*, 1278. [\[CrossRef\]](#)
26. Silvana, R.; Lv, J. Synchronization in time-sensitive networking: An introduction to IEEE Std 802.1. AS. *IEEE Commun. Stand. Mag.* **2023**, *6*, 14–20.
27. *IEEE Standard, 802.3-2022*; Ethernet, Single Balanced Twisted-Pair. IEEE Standard for Ethernet. Available online: <https://ieeexplore.ieee.org/document/988967/citations#citations> (accessed on 2 August 2025).
28. *IEEE Standard 802.1Q-2005*; IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks. IEEE: Piscataway, NJ, USA, 2006.
29. Zhao, L.; Liu, H.; Zhou, J. Design of Reliable Redundant Ethernet Protocols for Industrial and Automotive Applications. *IEEE Trans. Ind. Inform.* **2020**, *16*, 2453–2462.
30. *IEEE Std 802.1AS-2020*; IEEE 802.1 Time-Sensitive Networking Task Group. IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications. IEEE: Piscataway, NJ, USA, 2020.
31. Ambler, A.P. Economics of Design and Test. Realizing Complex System Design. In *Chapter in Book, Realizing Complex Integrated Systems*; CRC Press: Boca Raton, FL, USA, 2025; ISBN 9781003188377.
32. Yoon, G.Q.; Lee, S.G.; Kwon, D.H.; Kwon, S.C.; Park, Y.O. RAPIEnet based redundancy control system. In Proceedings of the 2011 11th International Conference on Control, Automation and Systems, Gyeonggi-do, Republic of Korea, 26–29 October 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 140–145.
33. *IEEE Std 802.3bw™-2015*; IEEE Standard for Ethernet—Amendment 5: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable. IEEE: Piscataway, NJ, USA, 2015.
34. Hua, K.A.; Abraham, J.A. Design of systems with concurrent error detection using software redundancy. In Proceedings of the 1986 ACM Fall Joint Computer Conference, Dallas, TX, USA, 2 November 1986; pp. 826–835.
35. Delong, T.A.; Johnson, B.W.; Profeta, J.A. A fault injection technique for VHDL behavioral-level models. *Des. Test Comput.* **1996**, *13*, 24–33. [\[CrossRef\]](#)
36. Rahmani, M.; Tappayuthpijarn, K.; Krebs, B.; Steinbach, E.; Bogenberger, R. Traffic Shaping for Resource—Efficient In-Vehicle Communication. *IEEE Trans. Ind. Inform.* **2009**, *5*, 414–428. [\[CrossRef\]](#)
37. Benso, A.; Prinetto, P.; Rebaudengo, M.; Reorda, M.S. A Fault Injection Environment for Microprocessor-based Boards” Politecnico di Torino Dipartimento di Automatica e Informatica Torino, Italy. In Proceedings of the International Test Conference, Washington, DC, USA, 18–23 October 1998; pp. 768–773.
38. An, W.; Médard, M.; Duffy, K.R. CRC codes as error correction codes. In Proceedings of the ICC 2021-IEEE International Conference on Communications, Montreal, QC, Canada, 14–23 June 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–6.
39. Lim, H.T.; Völker, L.; Herrscher, D. Challenges in a future IP/Ethernet-based in-car network for real-time applications. In Proceedings of the 48th Design Automation Conference, San Diego, CA, USA, 5–9 June 2011; pp. 7–12.
40. Lim, H.T.; Weckemann, K.; Herrscher, D. Performance study of an in-car switched ethernet network without prioritization. In *Proceedings of the Communication Technologies for Vehicles: Third International Workshop, Nets4Cars/Nets4Trains 2011, Oberpfaffenhofen, Germany, 23–24 March 2011*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 165–175.
41. Beak, S.; Hieu, B.; Lee, H.; Choi, S.; Kim, I.; Lee, K.; Lee, Y.; Jeong, T. Novel binary tree Huffman decoding algorithm and field programmable gate array implementation for terrestrial-digital multimedia broadcasting mobile handheld. *IET Sci. Meas. Technol.* **2012**, *6*, 527–532. [\[CrossRef\]](#)
42. Ballesteros, A.; Barranco, M.; Proenza, J.; Almeida, L.; Pozo, F.; Palmer-Rodríguez, P. An Infrastructure for Enabling Dynamic Fault Tolerance in Highly-Reliable Adaptive Distributed Embedded Systems Based on Switched Ethernet. *Sensors* **2022**, *22*, 7099. [\[CrossRef\]](#)
43. Danielis, P.; Skodzik, J.; Altmann, V.; Schweissguth, E.B.; Golatowski, F.; Timmermann, D.; Schacht, J. Survey on real-time communication via ethernet in industrial automation environments. In Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain, 16–19 September 2014; pp. 1–8.
44. An, W.; Médard, M.; Duffy, K.R. CRC codes as error correction codes. *arXiv* **2021**, arXiv:2104.13663.
45. Zhang, L.; Ye, S.; Gou, Z.; Yang, X.; Dai, Q.; Wang, F.; Lin, Y. An efficient parallel CRC computing method for high-bandwidth networks and FPGA implementation. *Electronics* **2024**, *13*, 4399. [\[CrossRef\]](#)
46. Koopman, P.; Chakravarty, T. Cyclic redundancy code (CRC) polynomial selection for embedded networks. In Proceedings of the International Conference on Dependable Systems and Networks, Florence, Italy, 28 June–1 July 2004; pp. 145–154.
47. NXP Semiconductors. MPC5553/554 Product Brief: Dual-Core e200z6-Based MCU for Automotive and Control Applications [Product Brief]. NXP. 2021. Available online: <https://www.nxp.com/products/MPC5553> (accessed on 2 August 2025).

48. Mentor Graphics. ModelSim Simulation and Verification Tool; Mentor, a Siemens Business: Wilsonville, OR, USA. Available online: <https://www.mentor.com/> (accessed on 2 August 2025).
49. Synopsys Inc. DesignWare Ethernet MAC IP User Guide. 2020. Available online: https://www.synopsys.com/dw/ipdir.php?ds=dwc_ether_mac10_100_universal (accessed on 2 August 2025).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.