Journal of Systems Architecture 169 (2025) 103605

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Time synchronization and performance analysis of the openSAFETY protocol via UDP over Ethernet☆

Shoaib Zafar *, Salvatore Sabina, Alessandro Biondi, Giorgio Buttazzo

Real-time Systems Laboratory, Scuola Superiore Sant'Anna, Pisa, Italy

ARTICLE INFO

Keywords:
Functional safety
OpenSAFETY protocol
Safety-critical systems
Real-time systems
Industrial IoT
Industry 5.0

ABSTRACT

The growing demand for Ethernet-based Industrial Internet of Things (IIoT) is changing the shape of modern industrial systems and emphasizing the need for high-speed, reliable, scalable, and safe communication among industrial devices. Ethernet-based networks provide the basis for seamless device integration, real-time data exchange, and increased operational efficiency, making them the key to Industry 5.0 applications. As industrial automation becomes increasingly complex, the importance of functional safety grows exponentially. The openSAFETY protocol is a fieldbus-independent, scalable, and robust protocol for implementing functional safety. Our contribution is twofold. First, we analyze time synchronization in the openSAFETY to fully understand the interrelated timing parameters and give some practical guidelines to tune the safety application. We have proposed the parameter tuning approach, which is better in terms of performance and ensures continuous, safe operations. Second, we analyze the protocol's performance via UDP over Ethernet under normal and degraded network conditions. We found the protocol resilient to network impairments under certain levels during the experiments. Under normal working conditions, the cycle time was successfully achieved in the microsecond range, even at full payload capacity.

1. Introduction

The Industrial IoT (IIoT) has revolutionized manufacturing and industrial processes by enabling connected devices, sensors, and systems to work together seamlessly, leading to increased efficiency, predictive maintenance, and optimized operations. However, as the complexity of these systems increases, the stakes for ensuring their reliable and safe operation increase. Functional safety is a cornerstone of Industry 5.0, which ensures the reliable operation of connected devices in environments where failures can lead to dangerous consequences [1].

Functional safety addresses risks from hardware failures, software failures, and system and communication failures through safe communication design, redundancy, and real-time error detection. In cyber–physical systems, where physical processes are tightly integrated with computation and communication, it is essential to have a synchronized and predictable behavior of machines exchanging safety-critical data. Ethernet-based IoT protocols are becoming a popular solution for industrial networking challenges because they offer significant advantages over traditional fieldbus standards. According to the HMS network survey report [2], in 2024, almost 71% of the industrial market is captured by devices using the Ethernet compared to the field bus, which is 22%. This shows the rapid increase in the demand for Ethernet-based solutions to achieve functional safety.

One possible way to achieve functional safety is to use "white channel communication" [3], in which every device used in the communication has a defined, predictable behavior. However, this comes at a cost because integrating the device increases the overall cost, and migrating to new technologies becomes slower. Another possible way is to use "black channel communication", which does not impose any restrictions on using the safety-integrated devices. However, maintaining the reliability and quality of the safety-critical data becomes challenging. Black channel communication sees the communication medium as a black box, and once the communication packet is sent to the network, it is unknown how much time it will take to be processed inside any device, which route it will take to reach the destination, or the integrity of the data if it has been corrupted. To address this challenge, the openSAFETY working group was established with Ethernet-Powerlink Group (EPSG) [4] to develop an open bus standard protocol for functional safety. They named this protocol as openSAFETY protocol. The openSAFETY protocol is the open bus standard that offers safe communication regardless of vendor and network.

☆ This paper is based on our understanding of the openSAFETY Protocol specification and implementation received from B&R automation, including the certified openSAFETY stack version 1.5.3 and specification version 1.5.2.

* Corresponding author.
E-mail address: shoaib.zafar@santannapisa.it (S. Zafar).

https://doi.org/10.1016/j.sysarc.2025.103605
Received 20 February 2025; Received in revised form 27 August 2025; Accepted 14 October 2025
Available online 22 October 2025

It seamlessly integrates with Ethernet/IP and other industrial protocols, reducing deployment costs and improving interoperability. The openSAFETY protocol achieves SIL 3 according to IEC 61508 [5] and can work on black channel communication. OpenSAFETY relies on a time synchronization mechanism to exchange safety-critical data. Successfully configuring a safety application for this data exchange requires a clear understanding and thorough analysis of the time synchronization mechanism used in the openSAFETY protocol.

So far, very limited work has been done on openSAFETY. Some authors [6,7] have performed the performance analysis of openSAFETY over MQTT via Wireless. Later, Hadžiaganović et al. [8] integrated the openSAFETY in OMNET++. Soury et al. [9], have discussed a case study of using the openSAFETY for the lift communication system. However, precise indications on how to set and tune protocol parameters, including those that are application dependent, are missing; we believe these indications are really important as they can directly affect the robustness, efficiency, and continuous openSAFETY operations. When parameters are correctly set and tuned, the communication between safety nodes will occur according to predictable timelines. An accurate tuning of synchronization parameters can prevent wasting resources from the overhead of redundant pre-processing of time synchronization messages, reducing the time to reach and maintain a time synchronization, and reducing the possibility of time synchronization failures during system operations. Well-tuned parameters guarantee a safe exchange of messages among openSAFETY nodes, even though the used communication network is not trustworthy in terms of real-time and safety requirements.

Our contribution is twofold. First, we analyze time synchronization in the openSAFETY protocol. Understanding the time synchronization mechanism is important for identifying the relationships between timing parameters and providing practical guidelines for effectively configuring safety applications. The key contribution of our work is the development of a structured approach for configuring safety applications. To the best of our knowledge, no established method currently provides clear and practical guidelines for the configuration of safety nodes to achieve accurate time synchronization. This paper provides justifications and discusses the impact of tuning the application parameters with experiments. Then, it evaluates the protocol's performance under normal and degraded network conditions. To assess the protocol's performance, two test cases have been designed:

- **Test Case 1:** Key performance metrics, such as latency, jitter, interframe delay variation, and bandwidth (when using UDP over Ethernet) are evaluated as a function of the payload size and transmission frequency of safety-critical data.
- **Test Case 2:** Network impairments are introduced to observe the protocol's behavior under non-ideal conditions. This test helped us understanding its resilience and performance when facing network disruptions.

2. Related work

Ethernet for Control Automation Technology (EtherCAT [10]) is a fieldbus system released by Beckhoff Automation. This protocol is based on the Master and slave-based communication model and uses the physical layer and standard frame defined in IEEE 802.3 standard [11]. The EtherCAT master sends a frame that passes through all of the slaves in the network that are connected and exchanges the data. The last connected slave detects the open port and returns the frame to the Master. Fail Safe over EtherCAT (FSoE) uses EtherCAT to achieve functional safety and is developed according to IEC 61508. Each FSoE device has its watchdog timer. If the FSoE-master does not receive a response from the slave till the watchdog timer times out expires, it triggers the safety conditions and puts the respective FSoE slave in the safe state. Although FSoE is still being used in the industry, FSoE is specifically designed for EtherCAT, which makes it difficult to integrate with other safety protocols used in different Fieldbus systems. EtherCAT relies on distributed clocks for time synchronization, with each connected EtherCAT slave equipped with a Distributed Clock (DC) chip. These distributed clocks provide nanosecond-level synchronization accuracy across all EtherCAT devices in the network. However, implementing distributed clocks requires specialized hardware support, which is a significant challenge and increases overall cost.

PROFInet protocol is the advanced form of Profibus [12] that allows communication between many FieldBus protocols that use the industrial Ethernet in compliance with international safety standards like IEC 61508 and ISO 13849 [13]. In PROFInet, real-time data transmission is based on cyclic data exchange. PROFIsafe is a safety protocol that is based on the PROFInet Protocol. To ensure a safe reaction time, the F-Devices (Safety Devices) use a watchdog timer that is restarted every time a new PROFIsafe message is received. Time synchronization in PROFInet IO is based on the precision transparent clock protocol (PTP) [14]. PROFIsafe is tightly coupled with PROFInet and PROFIbus systems, which limits its adoption in industries using other communication protocols [15]. CANopen Safety is a safety-critical communication protocol based on the CANopen framework [16], designed to meet the stringent requirements of functional safety applications [17] developed in compliance with IEC 61508 and other relevant safety standards. The basic concept of CANopen Safety is to transmit the safety-critical data in two independent messages. The first message contains the actual data, and in the second message, all data bits are inverted, with at least two bits inverted in the message identifier field. The safety-critical data is exchanged using SRDO (Safety Relevant Data Object) During the transmission, the timeout is monitored using two timeouts: SRVT (Safety-Relevant Validation Time) and Safeguard Cycle Time. SRVT is the maximum time allowed between the first and second message of an SRDO, and Safeguard Cycle Time is the time between multiple SRDOs, defining the maximum timeout between the occurrence of two consecutive SRDOs [18]. CAN openSAFETY relies on the CANopen protocol, which limits its adoption in industrial systems using other communication protocols. The openSAFETY protocol offers advantages over PROFIsafe, Fail Safe over EtherCAT, and CANopen Safety, especially in terms of flexibility, interoperability, and scalability. Unlike ProfiSAFE and Fail Safe over EtherCAT, which are closely tied to their specific parent fieldbus protocols, openSAFETY is independent of the underlying communication protocol, allowing compatibility with various industrial Ethernet and fieldbus systems. This independence enables integration across different networks, simplifies system architecture, and increases adaptability for various industrial applications.

The paper is structured to provide a detailed evaluation of network performance verification for the openSAFETY protocol, beginning with Section 3. This section presents the fundamental structure of openSAFETY communication models and frames, followed by an in-depth analysis of time synchronization. In this section, key definitions necessary for analyzing time synchronization are introduced. The robustness of time synchronization is subsequently explored. The time validation process is then examined, emphasizing its significance in maintaining the operational status of safety nodes and ensuring data reception. Section 4 discusses parameter tuning for time synchronization, where configuration parameters are tuned as part of the contribution. Section 5 evaluates the impact of tuning the application parameters on the safety application. Section 6 provides a performance analysis of the openSAFETY protocol. The evaluation is carried out under varying conditions. Test Case 1 focuses on the impact of different transmission frequencies and payload sizes, whereas Test Case 2 investigates network performance under impairments. Each test case is followed by a results section, which assesses key performance metrics such as propagation delay, latency, jitter, and interframe delay variation.

3. Network performance verification

The safety configuration manager (SCM) is the node responsible for managing all the safety nodes in a safety domain. Whenever a safety node is added or removed or becomes nonoperational, the safety configuration manager takes care of it. The main purpose of the SCM is to send periodic lifeguarding signals to the safety nodes in a safety domain. Standard openSAFETY has three types of frames, each used for different purposes:

1. Safety Network Management (SNMT).
2. Safety Service Data Object (SSDO).
3. Safety Process Data Object (SPDO).

The lifeguard signals are SNMT frames of the openSAFETY protocol. After becoming operational, each Safety Node must receive a lifeguard signal from the Safety Configuration Manager (SCM) within a specified duration, configured in the Safety Object Dictionary (SOD). The Safety Object Dictionary is a data structure that holds all parameters of a safety node, such as the unique identification number, lifeguarding, communication configurations, the consecutive time base, and more. Each node can have its own Safety Object Dictionary or be downloaded from the SCM to the safety nodes during initialization using SSDO frames. Safety-critical data is exchanged between safety nodes using SPDO frames. openSAFETY uses the producer and consumer communication model to exchange safety-critical data. The producer node broadcasts the data in the safety domain, and the consumer nodes with the producer node's SADR (Safety Address, a configurable parameter) can receive the data. This paper focuses on one of the most important aspects of the protocol, which is network performance verification, primarily involving SPDO frames. Therefore, it is assumed that the Safety Nodes (i.e., producer and consumer nodes) are operational and are receiving SNMT frames (i.e., lifeguarding signals) from the SCM. To cope with application requirements such as data freshness and data repetition, network performance verification needs to be done to determine the network's efficiency in meeting the requirements of the application. Network performance verification is carried out through two consecutive steps:

1. Time Synchronization;
2. Time Validation.

3.1. Time synchronization

Time synchronization is a process in which a consumer node updates itself about the relative time at the producer node. The consumer node is supposed to be the consumer of the safety-critical data sent by the producer node. Both the consumer and producer nodes can have different clock times (i.e., there may be a clock offset between any two nodes). However, to carry out successful time synchronization, the consecutive time-base (a configurable parameter) of both nodes (consumer and producer) should be the same. Time synchronization is done using the SPDO frames. There are three types of SPDO frames:

1. TReq: SPDO with a time request;
2. TRes: SPDO with a time response;
3. Data Only: SPDO with data-only.

An SPDO sent from the consumer node to the producer node for the time synchronization request is called the SPDO with time request or TReq. A consumer node sends one or more time synchronization requests (TReq) to its producer node to achieve time synchronization. Whenever a producer node receives a time request (TReq), it has to react immediately and send one or more time responses (TRes) back to the consumer node. The CT field of the openSAFETY frame contains information about the time instant at which the TReq or TRes is dispatched from the respective node.

After receiving the time response, the consumer node determines the TReq that was being answered. This information is stored in the TR field of the openSAFETY frame. Once the consumer node receives the TRes, it determines the round trip delay starting when the TReq is sent from the consumer node. Then, this delay will be checked against the constraints (see Definition 8 and 12). If the delay of the received TRes satisfies the constraints, then the time synchronization is said to be successful; otherwise, it is unsuccessful time synchronization. However, during normal operations, when time synchronization is done, the data is exchanged using SPDO data-only frames.

The openSAFETY protocol has a robustness feature, which allows sending multiple TReq and TRes consecutively, so if there is any loss in the transmission medium, the producer should receive at least one TReq. This analysis initially presents the time synchronization steps, assuming no lost messages exist (e.g., the protocol does not use the robustness feature). Then, the protocol stack feature (i.e., robustness feature) to cope with lost messages is described. To understand the time synchronization and time validation, the paper first examines some configurable parameters of both consumer and producer nodes. All configurable timing parameters in this paper use the consecutive time base (see Definition 1) as the base time unit.

Definition 1. The **Consecutive time base** is a configurable parameter of the consumer and producer nodes. The consecutive time base refers to the basic time unit (the Tick) used for time synchronization and time validation. The openSAFETY stack supports four different time bases (i.e., 1 μ s, 10 μ s, 100 μ s, 1 ms). The application designer can choose one of them depending on the application requirements and the efficiency (Scheduling and execution delays) of the safety nodes. This value should be equal for all of the safety nodes.

Definition 2. openSAFETY Transmit Process Data Object **(TxSPDO)** is the Tx or transmitter of the Safety Node (SN) responsible for sending the SPDO frame. openSAFETY Receive Process Data Object **(RxSPDO)** is the Rx or receiver of the Safety Node(SN) responsible for receiving the SPDO for time synchronization.

Definition 3. Refresh Prescale Consumer is the configurable parameter of the consumer node, which represents the delay between two consecutive SPDOs sent by the TxSPDO of the consumer node. It is denoted by Δt_c.

Definition 4. Refresh Prescale Producer is the configurable parameter of the producer node, which represents the delay between two consecutive SPDOs sent by the TxSPDO of the producer node. It is denoted by Δt_p. However, if new data is available at the producer node, SPDO can be sent to the consumer node without waiting for the refresh prescale timeout to expire.

In the openSAEFTY stack, TxSPDO and RxSPDO are the dedicated data structures responsible for sending and receiving the SPDO frames. For a Safety Node (SN), the RxSPDO is required if the time synchronization is needed to be done for that SN (i.e., if node is only producer node, then the node does not need to have a RxSPDO). However, at least one TxSPDO is mandatory for each SN; either it is the producer of the data (i.e., to send the SPDO data frames or sending the SPDO time response frames) or it is the consumer of the data (i.e., to send the time synchronization request to the producer node).

Definition 5. BestCaseTReqDelay(C) is a configurable parameter of the consumer node, representing the best estimation of the minimum time (from the consumer point of view) required for (a) transferring data (a time request (TReq)) from the consumer node to producer node and (b) to be processed and acknowledged by the producer node. This metric takes into account the optimal network performance conditions (i.e., network without any loss in transmission medium and

best transferring time) and the highest efficiency levels (minimum scheduling and execution delays) of the producer node. In this analysis, this parameter is denoted as β_C and formally defined in the equation below:

$$\beta_C = \min\left(\{dc_1, dc_2, dc_3, \ldots\}\right), \tag{1}$$

where dc_i represents the transmission delay of SPDO from the consumer node to the producer node in the ith observation of the experiment.

Definition 6. BestCaseTResDelay(P) represents the best estimation of the minimum time (from the producer point of view) required for transferring data (a time response(TRes)) from the producer node to the consumer node and its processing by the consumer node. This metric takes into account the optimal network performance conditions and the highest efficiency levels (minimum scheduling and execution delays) of the consumer node. In this analysis, this parameter is denoted as β_P and formally defined in the equation below:

$$\beta_P = \min\left(\{dp_1, dp_2, dp_3, \ldots\}\right), \tag{2}$$

where dp_i represents the transmission delay of SPDO from the producer node to the consumer node in the ith observation of the experiment.

Definition 7. BestCase minimum communication network round trip. This is the sum of BestCaseTReqDelay(C) and BestCaseTResDelay(P). In this analysis, we denote this parameter as β_{CP}.

Definition 8. Minimum TSync Propagation Delay is a configurable parameter of the consumer node, representing the minimum allowed time for a time response to be received by the consumer node as a reply to the sent time request. This delay is calculated from the moment the consumer node sends the time request and accounts for optimal network performance conditions and the highest efficiency levels of both the consumer and producer nodes. It establishes a time delay threshold below which the consumer node will not accept the response. In this analysis, this parameter is denoted as $\text{TSync}^{\text{min}}$.

Definition 9. Reaction Time (RT) is the maximum time for a single communication relationship (unidirectional) between the producer of the SPDO and the consumer of the corresponding SPDO. The time intervals required for preparing the SPDO and processing the SPDO are not included in this reaction time (i.e., Scheduling and execution delays). Therefore, it generally represents the maximum time for transferring the data frame from the sender node to the receiving node without considering the safety margin.

Definition 10. Safety Control Time (SCT): Safety Control Time (SCT) is an application-configurable parameter of the consumer node and represents the maximum allowed delay between the reception of two consecutive valid SPDOs at the consumer node, but only if the prior valid SPDO delay is equal to the reaction time. A failure should be raised if an SPDO is not received before the SCT delay. This parameter is also considered as the safety margin.

Definition 11. Safe Reaction Time (SRT) is the conservative value of the reaction time, including a safety margin equal to SCT. Therefore:

$$\text{SRT} = \text{RT+SCT}. \tag{3}$$

Definition 12. MaxTSyncPropagationDelay is the configurable parameter of the consumer node representing the maximum time allowed for a time response sent from a producer node to be received by the consumer node. This value includes the reaction time related to the TRes, and it is the estimation of the delay (from the consumer's point of view) which can be experienced in the reception of an SPDO because of the worst network conditions. For each TReq sent by the consumer node, this delay is computed as the time interval starting from the time instant when the TReq is sent.

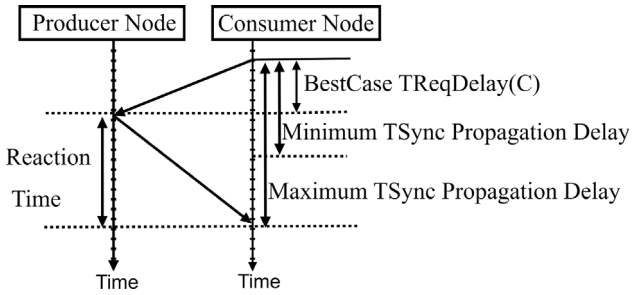

Fig. 1. Scenario where BestCaseTReqDelay(C) is the perfect estimation of the actual delay to send TReq and SPDO is received consuming the complete Reaction time (Successful Time Synchronization).

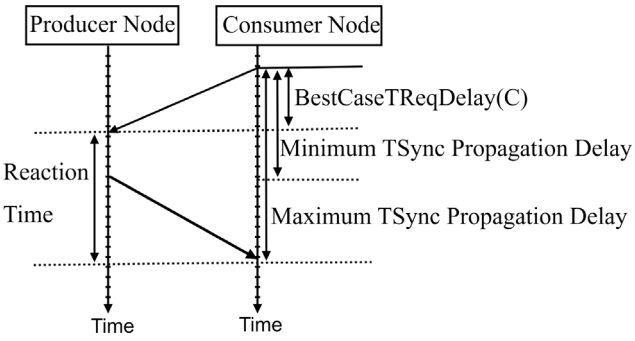

Fig. 2. Scenario where BestCaseTReqDelay(C) is the minimum best case estimation, and actual delay in sending TReq is more than that. In this case, the Reaction Time value includes this delay (Successful Time Synchronization).

Definition 13. Safe MaxTSync Propagation Delay is the conservative value of the MaxTSyncPropagationDelay, including a margin equal to SCT. This delay is calculated from the moment the consumer node sends the time request and accounts for worst network performance conditions and the lowest efficiency levels of both the consumer and producer nodes. It establishes a time delay threshold above which it is assumed that the consumer node will not have received the valid time response. This parameter is denoted as $\text{SafeTSync}^{\text{max}}$ and is defined as:

$$\text{SafeTSync}^{\text{max}} = \text{SCT} + \text{MaxTSyncPropagationDelay}. \tag{4}$$

We illustrate various scenarios of time synchronization in the context of estimating and handling delays for transmitting TReq and receiving TRes. In Fig. 1, the BestCaseTReqDelay(C) is perfectly estimated, and the TRes is received after fully consuming the reaction time, resulting in successful time synchronization. Fig. 2 shows a case where the BestCaseTReqDelay(C) represents a minimum best-case estimation, but the actual delay exceeds this value; in this case, the reaction time includes the extra delay, leading to successful synchronization. Fig. 3 depicts a situation where the BestCaseTReqDelay(C) is incorrectly estimated, causing the reaction time to fail in representing the maximum transfer time, resulting in unsuccessful time synchronization. Lastly, Fig. 4 presents a scenario where the BestCaseTReqDelay(C) is accurately estimated, and the TRes fully consumes the reaction time but is still received before the SRT timeout expires, achieving successful time synchronization.

Time synchronization is said to be successful if the Time Response (TRes) is received within the Minimum TSync Propagation Delay and Safe MaxTSync Propagation Delay window. If time synchronization is successful, then the consumer node memorizes the relative time of the producer node $\text{TRef}_{\text{Producer}}$ which is saved in the CT field of TRes and the $\text{TRef}_{\text{Consumer}}$ is defined below:

$$\text{TRef}_{\text{Consumer}} = \text{t+ BestCaseTReqDelay(C)}, \tag{5}$$

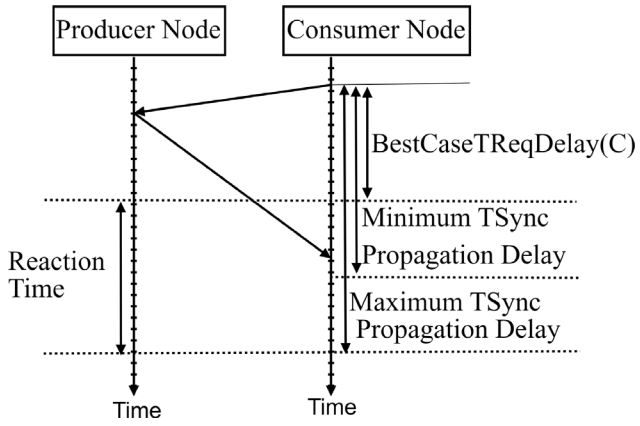

Fig. 3. Scenario where BestCaseTReqDelay(C) has been **wrongly** estimated. In this case, the reaction time value does not represent the maximum time for transferring the SPDO (Unsuccessful Time Synchronization).

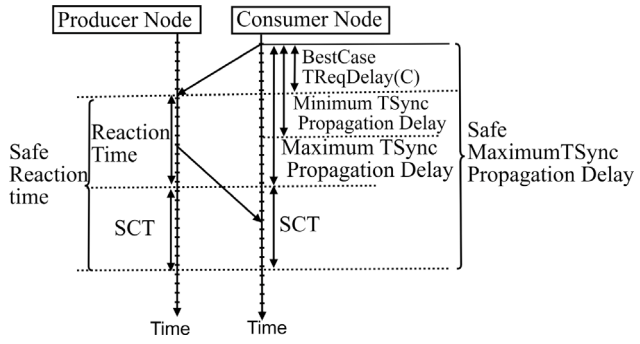

Fig. 4. Scenario where BestCaseTReqDelay(C) is the perfect estimation of the actual delay and TRes completely consumed the Reaction time but received before SRT timeout expires (Successful Time Synchronization).

where t is the time at which the time request (which has been answered) is sent. There is no need to inform the producer node about the successful time synchronization of the Consumer. There could be at most three cases if the time synchronization is unsuccessful.

1. Time Response Received Before the Minimum TSync Propagation Delay timeout expires.
2. Time Response Received After the Safe MaxTSync Propagation Delay elapses.
3. Time Response is not received due to loss in the transmission medium.

If the Time response is received before the Minimum TSync Propagation delay, then the Consumer will fall into FAIL SAFE STATE. It indicates that the BestCaseTReqDelay(C) has not been set correctly. If the Time response is received after the Safe MaxTSync Propagation Delay, then the consumer node will ignore the Time Response. There is no need to inform the producer node about the unsuccessful time synchronization of the Consumer.

3.2. Robustness in time synchronization

To increase the robustness of the protocol with respect to the loss of messages, the time request and time response are carried out as sequence of consecutive TReq and TRes. To study this feature, we will study some configurable parameters.

Definition 14. **Number of Consecutive Time Requests (*m*)** refers to the count of sequential time requests that a consumer node is configured to send to a producer node. This set is named a block of time requests. This configuration parameter allows the consumer node to send multiple time requests, interleaved by a time delay (refresh prescale Consumer, see Definition 3).

Definition 15. **Number of Consecutive Time Responses (*n*)** refers to the count of sequential time responses that a producer node is configured to send to a consumer node after the reception of the first-time request. The set of time responses is named a block of time responses. This configuration allows the producer node to send multiple time responses, interleaved by a time delay (refresh prescale producer, see Definition 4), whenever there is a need to send TRes back to the consumer node.

A consumer node can be configured to send multiple blocks of time requests to achieve the desired reliability of the communication. These blocks of time requests are interleaved by a time delay Td (see Definition 16). In each block of time request, the count of distinct time request numbers is stored as a TR counter within the TReq.

Definition 16. The consumer node sets the **Time Delay** after (a) having sent (*m*) time request(s) and (b) having waited for the Safe MaxTSync Propagation Delay from the last time request without receiving a valid time response. After this time delay, the consumer node is allowed to send (*m*) time request(s) for synchronization. Meanwhile, any time response received while Td has not elapsed will be ignored. This time delay is used when at least one time synchronization step is unsuccessful. It is denoted by Td.

Definition 17. During the time synchronization phase, **Time Request Cycle** is the maximum timeout from the start event of the synchronization phase, during which a consumer node has to receive a valid time response. Otherwise, a time synchronization failure must be raised when it expires. The management of this timeout includes the following stages:

1. **Initiation:** The time request cycle starts when the consumer node sends the first-time request to the producer node.
2. **Safe MaxTSync Propagation Delay Waiting:** After sending (*m*) time requests, the consumer node waits for Safe MaxTSync Propagation Delay from the last sent TReq.
3. **Time Delay (Td):** After sending the (*m*) TReqs, If no valid time response is received within the Safe maximum Tsync propagation delay of the Last TReq, the consumer node waits for time delay Td before sending another set of (*m*) time requests.
4. **Residual Refresh Prescale Timeout:** If the consumer node intends to send another TReq from a new block of *m* TReq(s), the consumer node will wait for the residual refresh prescale timeout after waiting for Td timeout. It is the remaining time in the refresh prescale to complete its duration after the Td timeout finishes. In this analysis, we have denoted it as α.
5. **Repetition:** The consumer node continues sending time requests and waiting for responses until a successful time synchronization is achieved or the Time Request Cycle timeout expires.

The time request cycle timeout is reset each time a successful time synchronization step is achieved. We have denoted this as **TRC**.

Consider the example in Fig. 5 in which a consumer node tries to establish the time synchronization with the Producer with (m=2, n=2, $\Delta t_p = \Delta t_c = 2$, $\beta_c = 3$, $\text{TSync}^{\min} = 4$, $\text{SafeTSync}^{\max} = 7$, Td=4) while the total blocks of time requests are two. From the first block of TReq, the first TReq is lost in a nonsafe communication medium, and second TReq is received successfully at the producer node. The Producer node starts to send time responses back to the consumer node in which the first TRes is lost and second TRes is received at the consumer node while the consumer node ignores this time response because this time

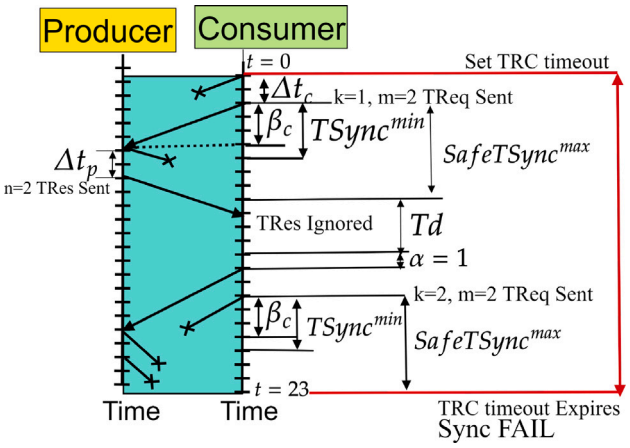

Fig. 5. Time Request Cycle timeout.

response is sent as a time response to the received time request (TR=2) from the first block of time request which has the SafeTSync^{max} = 7 and this timeout has already been passed. However, after passing the Td timeout, the consumer node cannot start another block of time synchronization because the refresh prescale timeout is not expired, which is taking the two time units in this example; therefore $\alpha = 1$ can be seen in Fig. 5. However, in this example, time synchronization cannot be achieved due to poor network conditions, and the time request cycle timeout elapses. Even though Time Synchronization is successful, in order to cope with clock drifts in producer and consumer Nodes and to keep the time synchronization error under a bounded value, a new synchronization phase is required after a certain time delay.

Definition 18. A Consumer Node sets **Time Delay Synchronization** when a successful time synchronization is reached after resetting the time request cycle. This parameter represents the maximum time delay between one successful time synchronization phase and the initiation of the next attempt to perform another time synchronization phase. It ensures that there is a controlled gap between successive time synchronization phases. When this timeout expires, a new time synchronization phase is started. It is denoted by **ts**.

Definition 19. Time To Synchronize (TTS) is defined as the total time taken by a consumer node to achieve successful time synchronization with a producer node. It begins at the moment the consumer node sends its first time synchronization request and ends when the consumer successfully synchronizes with the producer. The TTS may span one or more time request cycles, depending on parameter configuration and network conditions.

3.3. Time validation

The time validation phase starts after the successful time synchronization phase. The producer node does not need to be informed that the time validation phase has begun. The validation phase allows the consumer node to verify if the received SPDO meets the constraints on the propagation delay (Definition 20 21). After successful time synchronization, the consumer node is able to verify the quality of the received data. Whenever a successful time synchronization phase occurs, the consumer node resets the Time request cycle timeout (TRC) and sets the Time Delay Synchronization (ts) time out. To understand the time validation phase, we will first examine some parameters of a consumer node.

Definition 20. Minimum SPDO Propagation Delay is a configurable parameter of the consumer node, representing the minimum propagation delay for receiving a valid SPDO after successful time synchronization. This metric accounts for optimal network performance conditions and the highest efficiency levels of the consumer node. If the SPDO Propagation delay of the current received SPDO is less than the Minimum SPDO Propagation Delay, the consumer node will notify this anomaly and enter into a FAIL-SAFE state. We denote this parameter as SPDO^{min}.

Definition 21. Maximum allowed SPDO Propagation Delay: Represents the maximum allowable delay for receiving a valid SPDO after successful time synchronization. This metric accounts for the worst network performance conditions and the lowest efficiency level of the consumer node. If the consumer Node does not receive the SPDO after the Maximum allowed SPDO Propagation Delay timeout expires, the consumer node will ignore the SPDO. This parameter is equivalent to the Safe Reaction time. Maximum allowed SPDO Propagation Delay (SPDO^{max}) can be computed using the following formula:

$$\text{SPDO}^{max} = \text{SafeTSync}^{max} - \text{BestCaseTReqDelay(C)}. \tag{6}$$

TSPDO$_{Producer}$ is the timestamp of the producer node when the SPDO is dispatched. This value is included in the received SPDO. This timestamp serves as a reference for estimating the minimum SPDO propagation delay. While TSPDO$_{Consumer}$ is the time stamp of the consumer node at which SPDO is received.

Definition 22. SPDO Propagation Delay is the delay experienced in the reception of an SPDO at the consumer node with respect to the dispatch time of TSPDO contained in the received SPDO. We have denoted this as **PD** in our analysis. This delay is calculated using the following equation:

$$\begin{aligned}\text{PD} = &\left(\text{TSPDO}_{\text{Consumer}} - \text{TRef}_{\text{Consumer}}\right) \\ &- \left(\text{TSPDO}_{\text{Producer}} - \text{TRef}_{\text{Producer}}\right).\end{aligned} \tag{7}$$

Equation (7) stores the offset between the clocks of the Consumer and Producer Nodes at the time instant of successful time synchronization. However, since this offset typically changes over time due to the clock quality of both nodes, the protocol stack periodically forces new synchronization phases (see Definition 18). Without successful time synchronization, the Consumer Node cannot verify whether the SPDO propagation delay is correctly bounded.

Definition 23. Residual Safety Control Time (**SCT**$_R$) is the residual delay between the reception of two consecutive valid SPDOs at the consumer node after successful time synchronization. Each time a valid SPDO is received, the SCT$_R$ timeout is set. The next valid SPDO must be received before the SCT$_R$ timeout expires. If the SCT$_R$ timeout expires without the reception of a valid SPDO, the consumer node will enter into the FAIL-SAFE state. The value of SCT$_R$ timeout is set according to the following equation:

$$\text{SCT}_R = \text{SRT} - \text{PD}. \tag{8}$$

According to Definition 11, we have:

$$\begin{aligned}\text{SRT} = &\text{SCT} + \text{Maximum TSync Propagation delay} \\ &- \text{BestCaseTReqDelay(C)},\end{aligned} \tag{9}$$

However, during the operational phase of the protocol, the actual propagation delay, "SPDO Propagation Delay (see Definition 22)", can be less or greater than (Maximum TSync Propagation Delay - BestCaseTReqDelay(C)). The difference between (Maximum TSync Propagation

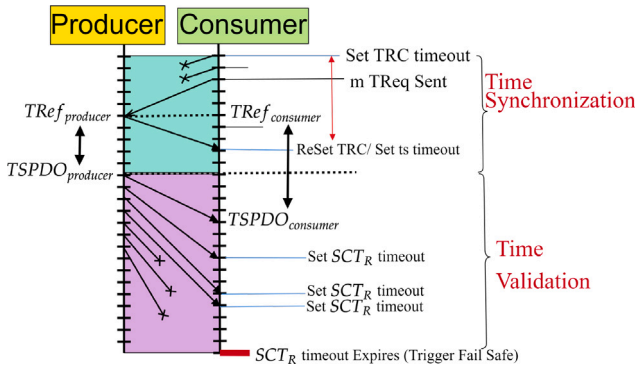

Fig. 6. Residual SCT timeout.

Delay - BestCaseTReqDelay(C)) and the actual propagation delay must be taken into account for setting the SCT_R, available for monitoring the reception of valid SPDO within an SCT time window. To this end, let us write the above equation of SRT as follows:

$$\text{SRT} = \text{SCT} + \big(\text{Maximum TSync Propagation Delay} - \text{BestCaseTReqDelay(C)} \big) - \text{PD} + \text{PD}. \tag{10}$$

Therefore, if we define the residual SCT as follows:

$$\text{SCT}_R = \text{SCT} + \big(\text{Maximum TSync Propagation Delay} - \text{BestCaseTReqDelay(C)} \big) - \text{PD}, \tag{11}$$

then we can write the above formula according to Definition 11 as follows:

$$\text{SRT} = \text{SCT}_R + \text{PD}, \tag{12}$$

which gives $\text{SCT}_R = \text{SRT} - \text{PD}$, as in Definition 23. Whenever the ts timeout expires, the consumer node sends another time synchronization request to the producer node and continues receiving the SPDO, checking its topicality by monitoring the SCT_R timeout. This can be seen in Fig. 7. In other words, it restarts the time synchronization phase, sets the Time Request Cycle timeout again, and keeps receiving the SPDOs. If it receives the TRes, it updates the $\text{TRef}_{\text{Producer}}$ and the $\text{TRef}_{\text{Consumer}}$, and sets the ts timeout again this can be seen in Fig. 7. If the SCT_R elapses, the consumer node will fall into the FAIL-SAFE state as shown in Fig. 6.

4. Parameters tuning for time synchronization

This section outlines some advantages of systematically tuning the openSAFETY parameters and application-related ones. Tuning such parameters according to the proposed approach reduces computation costs and ensures continuous, safe, and reliable operations in real-time environments. In the following subsections, we will explore the issues that arise from non-systematic parameter configurations and demonstrate how our approach addresses these challenges. The considerations discussed in these subsections do not impose any limitations on the application of the protocol stack.

4.1. Setting the no. of time requests (m) and time responses (n) in one block of time request cycle

Initially, when a consumer node starts the time synchronization, it enters a state in which it sends the first Time Request (TReq), and then it changes the state, waiting for the time response to be received, and continues to send the remaining TReq(s). When a producer node receives the first TReq, it sends a block of n TRes to the consumer node and any further TReq(s) will be ignored. The consumer node has to process the first received TRes and ignore the remaining TRes(s). Each received TRes requires protocol-level checks (e.g., to verify whether the received SPDO is valid for time synchronization, timestamp validation, etc.). If the transmission medium is highly reliable and there is less chance of loss of messages, then setting a high value of n for TRes should not be a good choice, as the consumer node has to pass protocol level checks for each received TRes. On the other hand, the producer node will also be busy sending the n TRes to the consumer node, which can cause a delay in serving another TReq (can be from another RxSPDO). On the other hand, the robustness feature of repetition of TReq and TRes is essential if there is a high loss in the transmission medium, and setting the low value of m and n can increase the risk of synchronization failure.

The proposed method determines the best configuration of m and n based on the probability of loss of messages on the transmission medium. It models the probability of message loss and the desired probability of successful synchronization to estimate the number of m and n that systematically balance the redundancy and efficiency. To model this, we consider BER as the probability that a single bit will be received in error. Forward error correction (FEC) mechanisms can be applied to the physical layer to address bit errors. However, these techniques cannot correct all possible bit errors within a frame [19]. For this reason, we refer to the residual BER as the P_e after redundancy has been applied for error correction. In this context, the transmission of each bit in a frame can be treated as an independent Bernoulli trial, where each trial (bit) can either succeed (be transmitted correctly) with probability $1 - P_e$ or fail (not transmitted correctly) with probability P_e which represents the probability that a single bit is received in error or lost due to noise or other impairments in the transmission medium. This analysis assumes that the bit error rate (BER) remains uniform across all bits in a frame. For successful time synchronization, all L_{TReq} bits in the frame must be received correctly without any loss. The probability of successfully transmitting a frame of L_{TReq} bits, denoted as $P_s(\text{TReq})$, is given by:

$$P_s(\text{TReq}) = (1 - P_e)^{L_{\text{TReq}}}. \tag{13}$$

When the consumer node sends (m) TReqs, the goal is to ensure that the producer node receives at least one TReq successfully. The probability of failure (not being received) for a single TReq is $1 - P_s(\text{TReq})$, and the probability of all (m) TReqs failed to receive is $(1 - P_s(\text{TReq}))^m$. The complement of this gives the probability that the producer node receives at least one TReq, which can be derived using the Complement rule of probability as follows:

$$P_{m,\text{success}} = 1 - (1 - P_s(\text{TReq}))^m. \tag{14}$$

Similarly, under the assumption that each Time Response (TRes) is affected by random errors only, the probability that the Consumer successfully receives at least one of the (n) time responses is:

$$P_{n,\text{success}} = 1 - (1 - P_s(\text{TRes}))^n. \tag{15}$$

P_{failure} is the probability of failure that (m) TReq and (n) TRes frames failed to be received by their respective nodes in one block of time synchronization and defined as follows:

$$P_{\text{failure}} = (1 - P_s(\text{TReq}))^m \times (1 - P_s(\text{TRes}))^n. \tag{16}$$

Let us assume P_{block} is the desired probability that the consumer node successfully synchronizes with the producer node in one block of the Time request cycle. This can be computed as the probability of receiving at least one correct time response in the consumer node, conditioned to the reception of one correct time request in the producer node. This probability P_{block} can be computed as:

$$P_{\text{block}} = P_{m,\text{success}} \times P_{n,\text{success}}. \tag{17}$$

We can define the communication cost (here referred to as Cost) in terms of transferred bits between the consumer and producer node in one block of the time request cycle:

$$\text{Cost} = m \times L_{\text{TReq}} + n \times L_{\text{TRes}}. \tag{18}$$

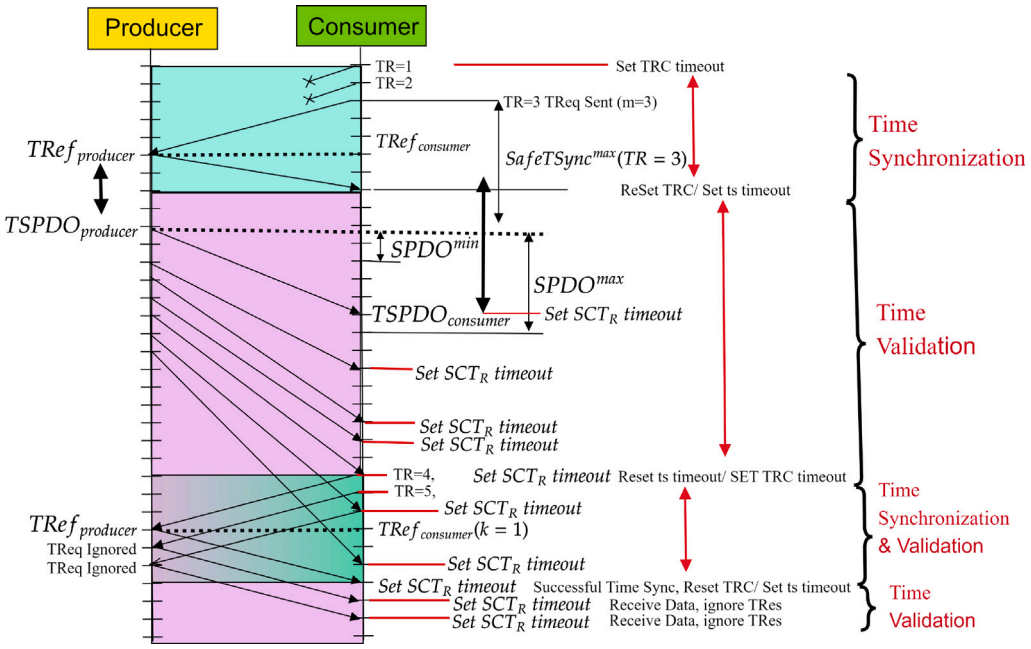

Fig. 7. Successful Time Synchronization and Time validation.

It is important to note that the openSAFETY stack under analysis supports a maximum of 63 consecutive time requests, denoted as $m_{\max}$, and for time responses, the variable is an 8-bit unsigned integer, allowing a maximum of 255 time responses, denoted as $n_{\max}$. Thus, we can iteratively search for the values of m and n using Algorithm 1 to minimize the communication cost. By modeling the frame receiving/transmitting success probabilities under bit error rates (BER) and iteratively searching for the smallest pair (m, n) that satisfies a target reliability threshold, the proposed approach can systematically balance redundancy (to counteract frame loss) and efficiency (to reduce unnecessary pre-processing).

Algorithm 1 Determine Configuration of Numbers of Time Requests and Time Responses.

Require: Maximum $m = m_{\max}$, Maximum $n = n_{\max}$
Require: P_e, P_{block}, L_{TRes}, L_{TReq}
Ensure: Best configuration of m and n
Compute $P_s(\text{TReq}) \leftarrow (1 - P_e)^{L_{\text{TReq}}}$
Compute $P_s(\text{TRes}) \leftarrow (1 - P_e)^{L_{\text{TRes}}}$
$P_{\text{F\_threshold}} \leftarrow 1 - P_{\text{block}}$ ▷ Failure threshold
Initialize min_cost $\leftarrow \infty$, best_config $\leftarrow \emptyset$
for $m = 1$ to $m_{\max}$ **do**
  for $n = 1$ to $n_{\max}$ **do**
    Compute $P_{\text{failure}} \leftarrow (1 - P_s(\text{TReq}))^m \times (1 - P_s(\text{TRes}))^n$
    if $P_{\text{failure}} \leq P_{\text{F\_threshold}}$ **then**
      Compute Cost $\leftarrow m \times L_{\text{TReq}} + n \times L_{\text{TRes}}$
      if Cost < min_cost **then**
        min_cost $\leftarrow$ Cost
        best_config $\leftarrow (m, n, P_{\text{failure}}, \text{Cost})$
      end if
    end if
  end for
end for
return best_config

4.2. Setting the time delay timeout (td)

The time delay Td introduces a controlled time gap between two consecutive blocks of TReqs sent by a consumer node. When such a delay is well dimensioned, this controlled gap enables the following two benefits: (A) it allows the consumer node to potentially wait for receiving all the block of TRes before sending a new set of TReq(s), and (B) it prevents the consumer node from sending a TReq during a time interval when the communication channel is experiencing a burst of errors [20]. Consider the example shown in Fig. 8. The consumer node sends TReqs (represented in blue) to the producer node over a communication medium prone to high message loss. In this case, the first TReq is lost, but the second TReq is successfully received by the producer node (indicated by the filled blue arrowhead). The producer node then begins sending time responses (TRes) back to the consumer node (shown in purple). We can see TRes is received by the consumer node but ignored due to violating the constraints (represented as a bar at the end of the arrowhead).

If Td is too short, the consumer node sends another set of m TReq(s) before all TRes associated with previously received TReq are fully processed. If the producer node is still responding to the previous TReq, it cannot process the new TReq, even if it is successfully received. This situation can introduce processing overheads for both the producer node discarding these new TReqs, and the consumer node processing useless TRes, which will be discarded. These issues can cause unexpected delays in synchronization. On the other hand, setting Td too long is also undesirable because it unnecessarily increases the overall synchronization time. To set the Td delay using the proposed systematic approach, two key channel properties have been taken into account: (A) the channel property with respect to bursts of errors, and (B) the channel property with respect to the random message loss.

In cases of burst errors, as described in [20], the consumer node should ideally wait until the Burst has subsided before sending another request. This is important because, during the Burst, any sent TReq(s) would likely be ineffective or affected by common-mode errors. Burst error durations are defined by standardized hardware tests for electromagnetic compatibility (EMC) as per EN 61000-4-4 [21], with pulse patterns outlined in Clause 6.2.2. Therefore, the consumer node must consider the maximum burst duration when determining the Td delay.

Let us consider a situation where the channel experiences random message loss. In Fig. 9, Just before the SafeTSync$^{\max}$ timeout expires, the producer node receives a TReq and begins sending n TRes to the consumer node. As a result, the producer node won't be able to respond to any other TReq for $(n-1)\times\Delta t_p$, and all the TRes sent during this time will be ignored by the consumer node. This is because the consumer node has already waited for SafeTSync$^{\max}$ from the last TReq. After

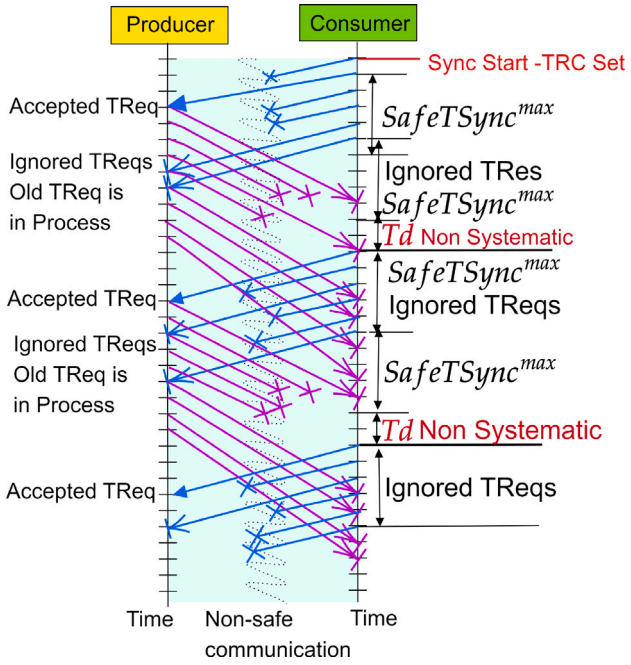

Fig. 8. Non systematic Td timeout.

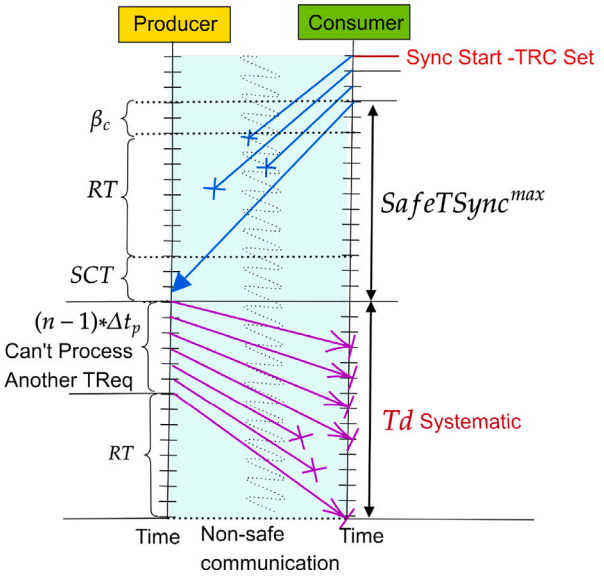

Fig. 9. Systematic Td Timeout.

waiting for $(n-1)\times\Delta t_p$, if the consumer node waits for the reaction time (RT) of the last TRes sent, it ensures that the maximum number of TRes will be discarded during the Td timeout. This minimizes the chance of receiving unwanted TRes (i.e., from the previous block of TRes) when a new block of TReq is initiated. Therefore, considering the channel's key properties, the consumer node should wait for the maximum time between $\left((n-1)\times\Delta t_p + \mathrm{RT}\right)$ and the Burst Error Duration before starting a new block of TReq. The proposed approach suggests to adjust the Td as follows:

$$\mathrm{Td} = \max\left((n-1)\times\Delta t_p + \mathrm{RT}, \text{ Burst Error Duration} \right). \tag{19}$$

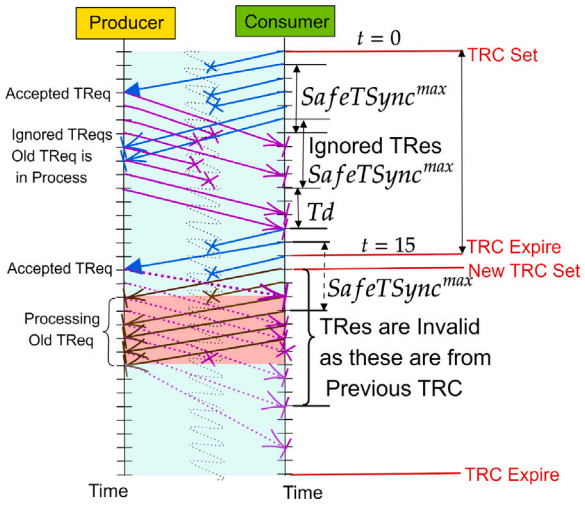

Fig. 10. Non systematic TRC timeout.

4.3. Setting the time request cycle (TRC)

The Time Request Cycle is an essential parameter during time synchronization, where the consumer node attempts to synchronizes with the producer node. A non-systematic configuration of the TRC can increase the TTS (See Definition 19), especially when there is a high probability of message loss in the transmission medium.

Consider an example of non-safe communication in Fig. 10 with parameters $\Delta t_c = \Delta t_p = 1$, $\mathrm{SafeTSync}^{\max} = 5, \mathrm{Td} = 3, \mathrm{TRC} = 15, m = 6$, $n = 8$. As the consumer node sets the TRC, the first TReq is lost. The second TReq is received (with a blue arrowhead), and the producer node sends the TRes back to the consumer node. All TRes (shown in purple) are lost or arrived after passing the $\mathrm{SafeTSync}^{\max}$, so these TReqs will be ignored (can be seen as the bar on the arrowhead). After the Td timeout, the consumer node starts sending another set of m TReq(s); in this example, the first new TReq is sent at time $t = 14$ and it has been successfully received by the producer node. According to the protocol, the producer node starts sending the n TRes to the consumer node (it can be seen in purple dotted arrows). Based on the configured TRC, the TRC timeout expires at time $t = 15$, and the consumer node declares the time synchronization failure. When the synchronization failure occurs, after passing the Δt_c, the openSAFETY starts a new time synchronization phase, sets a new TRC, and sends a new TReq. The producer node successfully receives these new TReqs (shown in brown), but these new TReqs cannot be processed because the producer node is still busy in responding to the TReq accepted at time $t = 14$ (shown in dotted arrows). According to the openSAFETY, if the producer receives a new TReq and is already responding to the same RxSPDO, it will ignore the new TReq. So, these TReqs will be ignored, and all of the received TRes at the consumer node are invalid as these are from previous TRC for which the synchronization failure has already been declared. OpenSAFETY stack generates the error(s) whenever it receives invalid TRes(s). We can see in Fig. 10 that, although at $t = 18$ the consumer node receives the TRes within the $\mathrm{SafeTSync}^{\max}$ window, this TRes is invalid as the consumer node has already declared the Sync-fail for the TRC during which this TReq was sent. Fig. 10 shows the time in the red-shaded region when a producer node ignores the new TReq(s) as it is busy processing old TReq, and the consumer node receives the invalid TRes(s).

The proposed solution to address this problem is to define the time request cycle in k blocks ($k = 1, 2, \ldots$) of TReq(s) such that, whenever the TRC timeout expires, the consumer node has already passed the Td timeout without sending another TReq. If a block of TReq is added to the time request cycle, it is mandatory to exactly complete its entire

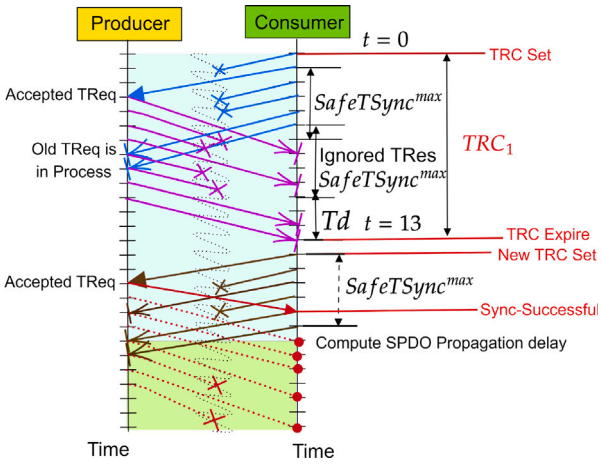

Fig. 11. Systematic TRC Timeout.

duration, including the Initiation phase, Safe MaxTSync Propagation Delay Waiting, Time Delay, and Residual Refresh Prescale Timeout (see the first four points under the management of TRC timeout in Definition 17). We can determine the minimum number of such k blocks in a TRC based on the target probability of successful synchronization.

The probability P_{block} of success of the time synchronization process in a single block is given by Eq. (17). From this value, we can compute the probability of failure in one block of the time request cycle: $P_{\text{block\_fail}} = 1 - P_{\text{block}}$. As each synchronization block is independent of the previous one, the probability of synchronization failure after k blocks ($k = 1, 2, \ldots$) of TRC is as follows:

$$P_{\text{fail}}(k) = (P_{\text{block\_fail}})^k. \tag{20}$$

The desired reliability (R) of the transmission medium is the target probability of accomplishing synchronization in k blocks of TRC, i.e., $P_{\text{fail}}(k) \leq 1 - R$. Given R, we can compute the minimum amount of blocks required to achieve the desired reliability by rearranging the last inequality, hence obtaining:

$$k \geq \left\lceil \frac{\log(1-R)}{\log(P_{\text{block\_fail}})} \right\rceil. \tag{21}$$

The overall duration of time request cycle comprise of k block is represented as TRC_k in Eq. (22) . Two time requests are delayed by Δt_c in a single block of time requests. To send m time requests, the maximum time will be $(m-1) \times \Delta t_c$. After sending the last time request (TReq), there will be a waiting period of $\text{SafeTSync}^{\max}$. If the consumer node intends to start another time request, it must wait for the duration Td. However, after waiting for Td, the consumer node may still be unable to send the time request again due to the Refresh prescale Consumer has not expired (denoted as α in Eq. (22)). Based on the protocol specification, the time request for generic k blocks ($k = 1, 2, \ldots$) can be written as:

$$\text{TRC}_k = k\left[(m-1)\cdot \Delta t_c + \text{SafeTSync}^{\max}\right] + k[\text{Td} + \alpha], \tag{22}$$

where α is the residual Refresh prescale timeout consumer and is defined as:

$$\alpha = \begin{cases} 0, & \text{if } (\text{TW}_c \mod \Delta t_c) = 0; \\ \Delta t_c - (\text{TW}_c \mod \Delta t_c), & \text{otherwise,} \end{cases} \tag{23}$$

where TW_c (Time Wait Consumer) is defined as follows:

$$\text{TW}_c = \text{SafeTSync}^{\max} + \text{Td}. \tag{24}$$

Suppose we consider the same scenario discussed in Fig. 10, with identical parameter configurations except for the TRC timeout, which is now set to $\text{TRC}_1 = 13$ according to Equation (22), we observe a better outcome as shown in Fig. 11, the TRC timeout expires at time $t = 13$, prompting the consumer node to set a new TRC at $t = 14$ and send a new TReq (indicated by the brown arrowhead). The corresponding TRes is received and processed (indicated by the red arrowhead), resulting in successful synchronization (illustrated by the green shaded region with a red circle marking the accepted and processed SPDO after synchronization), which did not occur in the previous case. In this scenario, synchronization is successfully achieved at time $t = 18$ during the second TRC, unlike the earlier scenario. Our proposed approach helps in reducing the time to synchronize (TTS).

4.4. Setting the time synchronization timeout (ts)

After successful time synchronization, the consumer node stores the values of $\text{TRef}_{\text{Consumer}}$ and $\text{TRef}_{\text{Producer}}$ (as defined in Equation (7)). However, this offset changes over time due to the tolerance of the quartz crystal, which can cause the clocks to drift apart. Generally, this clock drift (λ) can be either positive or negative. The clock drift can cause the consumer node to:

1. Calculate an SPDO propagation delay that increases continuously during time validation.
2. Calculate an SPDO propagation delay that decreases continuously during time validation.

If the consumer node calculates the SPDO propagation delay that increases over time, it can cause the SPDO propagation delay to be greater than the residual safety control time. According to the definition 23 of the residual safety control time, we can write SCT_R = (SCT-PD)+RT. This relationship leads to the fact that to keep the consumer node operational and continue receiving the SPDOs, the SCT-PD $\geq$ 0. Otherwise, the residual safety control time will be less than the reaction time, which cannot be possible to keep the safety node operational. This leads to the fact that:

$$\begin{aligned} &\left(\text{TSPDO}_{\text{Consumer}} - \text{TRef}_{\text{Consumer}}\right) \\ &- \left(\text{TSPDO}_{\text{Producer}} - \text{TRef}_{\text{Producer}}\right) \leq \text{SCT}. \end{aligned} \tag{25}$$

Consider the following nominal values during the time validation:

$$\begin{aligned} \text{TSPDO}^N_{\text{Consumer}} &= \text{Nominal value of TSPDO}_{\text{Consumer}}. \\ \text{TSPDO}^N_{\text{Producer}} &= \text{Nominal value of TSPDO}_{\text{Producer}}. \end{aligned}$$

We can write Equation (25) by incorporating the clock drift accumulated over time:

$$\begin{aligned} &\left(\text{TSPDO}^N_{\text{Consumer}} \pm \lambda t - \text{TRef}_{\text{Consumer}}\right) \\ &- \left(\text{TSPDO}^N_{\text{Producer}} \pm \lambda t - \text{TRef}_{\text{Producer}}\right) \leq \text{SCT}. \end{aligned} \tag{26}$$

$$\begin{aligned} \implies |t| \leq{}& \frac{\text{SCT} - \left(\text{TSPDO}^N_{\text{Consumer}} - \text{TRef}_{\text{Consumer}}\right)}{|\pm 2\lambda|} \\ &+ \frac{\left(\text{TSPDO}^N_{\text{Producer}} - \text{TRef}_{\text{Producer}}\right)}{|\pm 2\lambda|}. \end{aligned} \tag{27}$$

The above equation ($|t| = \text{ts}(1)$ as first consideration) can be simplified, assuming the consumer and producer nodes have the same clock drifts:

$$\text{ts}(1) \leq \frac{\text{SCT} - \text{Reaction Time}}{|\pm 2\lambda|}. \tag{28}$$

We can simplify the above equation by considering the clock drifts of the Producer and Consumer node separately as follows:

$$\text{ts}(1) \leq \frac{\text{SCT} - \text{Reaction Time}}{|\pm \lambda_{\text{Consumer}} \pm \lambda_{\text{Producer}}|}. \tag{29}$$

While $\pm\lambda_{\text{Consumer}}$ and $\pm\lambda_{\text{Producer}}$ are the clock drifts of the consumer and producer nodes, respectively.

If the consumer node calculates the SPDO propagation delay that decreases over time, then it can cause the SPDO propagation delay to

be less than $SPDO^{min}$. Therefore, according to the definition 20, the following condition should be valid during the time validation:

$$\left(TSPDO^{N}_{Consumer} \pm \lambda t - TRef_{Consumer}\right) - \left(TSPDO^{N}_{Producer} \pm \lambda t - TRef_{Producer}\right) > SPDO^{min}. \quad (30)$$

During the time validation, the consumer node have additional time SCT as safety margin.

$$\left(TSPDO^{N}_{Consumer} \pm \lambda t - TRef_{Consumer}\right) - \left(TSPDO^{N}_{Producer} \pm \lambda t - TRef_{Producer}\right) + SCT > SPDO^{min}. \quad (31)$$

The above equation ($|t| = ts(2)$ as second consideration) can be simplified as follows:

$$ts(2) \leq \frac{SRT - SPDO^{min}}{|\pm\lambda_{Consumer} \pm \lambda_{Producer}|}. \quad (32)$$

The maximum value of ts (ts^{max}) can be chosen as the minimum value between ts(1) and ts(2):

$$ts^{max} = \min\left(ts(1), ts(2)\right). \quad (33)$$

Equation (33) represents the maximum value of ts during which time synchronization must happen otherwise the consumer node will fall into FAIL-SAFE due to increase in the synchronization error. The maximum estimation of the synchronization error is dependent of the ts^{max} also on and static delays (D_{Static}) (NIC Latency,scheduling and execution delays) and maximum jitter (J_{max}). The maximum synchronization error (E^{max}) can be given by the following equation:

$$E^{max} = |\pm\lambda_{Consumer} \pm \lambda_{Producer}| \times ts^{max} + J_{max} + D_{Static}. \quad (34)$$

4.5. Setting the configuration parameters for time synchronization

In this section, we summarize the key configuration steps needed to achieve successful time synchronization, as previously discussed. Application designers can follow these steps to properly adjust the TxSPDO and RxSPDO parameters as required.

1. **RxSPDO:** Set Td according to the Equation (19)
2. **RxSPDO:** Set *m,n* using Algorithm 1.
3. Estimate k according to the Equation (21).
4. **RxSPDO:** Set Delay $SPDO^{max} = SRT$.
5. **RxSPDO:** Set SCT = SRT - RT.
6. **RxSPDO:** Set MaxTSyncPropDelay = $\left(SRT\text{-}SCT + \beta_c\right)$.
7. $SafeTSync^{max} \leftarrow MaxTSyncPropDelay + SCT$.
8. **RxSPDO:** Set $\beta_c < TSync^{min} < \beta_{cp}$.
9. **RxSPDO:** Set Delay $SPDO^{min} < \beta_p$.
10. **RxSPDO:** Compute and set TRC_k according to Eq. (22).
11. **RxSPDO:** Set $ts < ts^{max}$.

5. Experimental analysis on the tuning of some key parameters

This section highlights the positive impact of parameter configurations based on the proposed approach, referred to as Systematic Configuration, compared to configurations following alternative methods, which we categorize as Non-Systematic Configuration. In the context of this paper, the following three configuration parameters have been analyzed: TRC, ts, and (m, n). The used experimental setup involves three safety nodes within a dedicated Local Area Network (LAN) connected via Ethernet. These nodes include the Safety Control Manager (SCM), the SN–Producer Node, and the SN–Consumer Node. The SCM is responsible for sharing the Safety Network Management Telegram (SNMT) and Safety Service Device Object (SSDO) frames, while the SN–Producer Node handles the transmission of Safety-Process Data Object (SPDO) frames from the producer to the consumer node. The Safety Hardware Near Firmware (SHNF), utilizes the UDP protocol for communication. The experimental setup is shown in Fig. 12, and detailed information about the safety nodes can be found in Table 8.

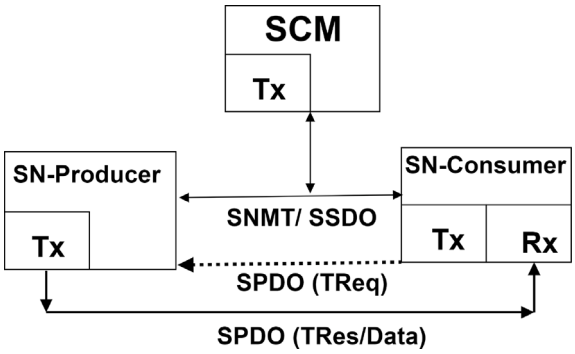

Fig. 12. Experiment setup.

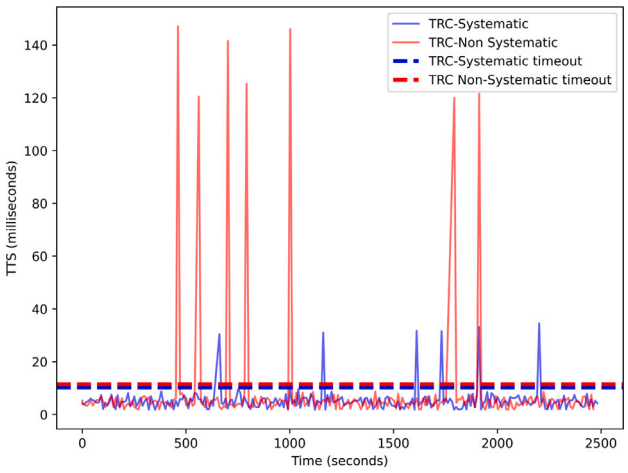

Fig. 13. Experimental Results of tuning TRC.

5.1. Experimental analysis of TRC tuning

For evaluating the impact of a well-configured TRC parameter, we have introduced the controlled network impairments, NETEM [22], shown in Table 2. The introduced network impairments are delay, jitter, maximum packet loss, and the correlation of error burst (CRERR), which shows the likelihood of losing another frame if one frame is lost. The protocol and application parameters used to perform this analysis are shown in Table 1. This experiment aims to determine the time to synchronize (TTS) (See Definition 19) over time, and shows the advantages obtained when TRC is set according to our approach. See Table 3 for the Systematic and Non-Systematic TRC timeout configurations used in these experiments.

The experiment results are shown in Fig. 13, where the x-axis represents the time in seconds, while the y-axis shows the TTS in milliseconds associated with both configurations. The red and blue lines represent the TTS for non-systematic and systematic TRC configurations, respectively; the horizontal dotted lines show the configured TRC values. The red line corresponds to the Non-Systematic TRC, and the blue line corresponds to the Systematic TRC timeout. TTS mostly remains below 10 ms for both configurations, indicating that the synchronization between the consumer and producer nodes is on average, efficient. However, TTS values are strongly different when TRC elapses. With the non-systematic TRC value, TTS can reach spikes up to 143 ms. When TRC elapses, after passing the refresh prescale consumer, the consumer node attempts to synchronize again but often receives invalid time response(s) from the producer node, and the producer node remains busy in sending the TRes due to previously accepted TReq. This can lead to synchronization failures, and the consumer node must

Table 1
Fixed application parameters.

Parameter	SCT	RT	m	n	Δt_c	Δt_p	Td	Payload	ts	$SPDO^{min}$	SNMT timeout	Test duration
Value	20 (ms)	3 (ms)	3	5	0.3 (ms)	1 (ms)	4 (ms)	311 Bytes	10 (s)	0.2 ms	1 (s)	2500 (s)

Table 2
Introduced network impairments.

Test name	Delay (ms)	Jitter (ms)	Max Packet Loss	CRERR
TTS	5	± 3	5%	15%

Table 3
Systematic and non-systematic TRC timeouts.

Test name	Systematic TRC (k=1)	Non-Systematic TRC (1<k<2)
TTS	27.9 (ms)	28.5 (ms)

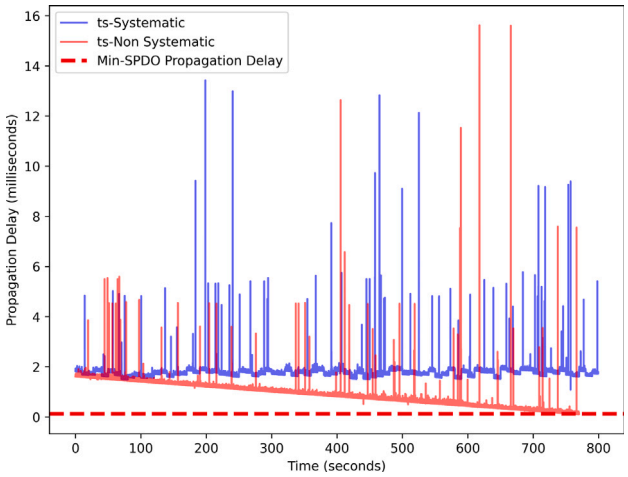

Fig. 14. Experimental results of tuning ts.

repeatedly attempt to synchronize, which can cause unexpected delays. Whereas, with the systematic TRC value, the highest spike is around 35 ms. The experimental results provide strong evidence supporting the effectiveness of the proposed Systematic TRC approach.

5.2. Experimental analysis of ts tuning

To conduct the experiment on the impact of the ts timeout, we first measured the clock drifts of the producer and consumer nodes using the Chrony tool on Linux [23]. Over the course of the experiment, we collected more than 1,000 clock drift observations and used the maximum drift value as the approximated clock drift for the safety node. In this case, the accumulated clock drifts of both the producer and consumer nodes were approximately a negative 30 ppm with a maximum skew of ±1.2 ppm. We then ran the test using the parameters described in Table 1, with the only variation being the ts timeout. Table 4 reports the ts^{max} derived from the Equation (25) and the Systematic and Non-Systematic ts timeout values used in the test; note that our approach allows us to compute the maximum ts and, thus, we choose a ts less than such a maximum value, e.g. 10 s. On the other hand, in the case of a non-systematic ts timeout value, the application designer does not know in advance that a ts = 800(s) is an incorrect parameter that may lead to unavailability or, worst, safety issues if the minimum SPDO propagation delay is incorrectly configured and the clock drift of both nodes is negative.

There were no induced network impairments in these experiments. The experiment results are presented in Fig. 14 where the SPDO Propagation Delay is plot over time for the two different configurations. The

Table 4
Systematic and non-systematic ts timeout.

Test name	ts^{max}	Systematic ts ($ts< ts^{max}$)	Non-Systematic ts ($ts> ts^{max}$)
ts-test	760 (s)	10 (s)	800 (s)

Table 5
Application specifications: Best configuration test for m and n.

L_{TReq}	L_{TRes}	P_{block}	ts	TRC	Duration
55-Bytes	57-Bytes	0.96	10 (s)	43.2 (ms)	5 min

x-axis shows the time in seconds, and the y-axis shows the propagation delay in milliseconds. The blue line corresponds to the experiment with the Systematic ts configuration, while the red line represents the experiment with the Non-Systematic ts configuration. The horizontal dashed red line indicates the minimum allowed SPDO propagation delay, which serves as the lower limit for the acceptable propagation delay.

The purpose of this experiment was to show the impact of using a wrong ts timeout, greater than ts^{max} computed using our approach. The Non-Systematic ts configuration (red line), has a negative trend due to the clock drift between the producer and consumer nodes. As the measured clock drifts of both nodes were negative, such negative drifts lead to a decreasing SPDO propagation delay calculation over time. When such a propagation delay drops below the minimum allowable threshold, which is represented by the red dashed line, the consumer node enters the FAIL-SAFE state. This occurs when t is around 770 s. On the other hand, using our approach, the maximum allowed ts value is computed, i.e. 760 s, and the application designer can set the ts parameter to a value that will not lead a consumer node to fall into the FAIL-SAFE state.

5.3. Experimental analysis on m and n

To evaluate the impact of configuring the number of time requests (m) and time responses (n) on the overall efficiency of the time synchronization protocol, we conducted 5 min experiments under varying levels of randomly induced frame loss, ranging from 0% to 4% (See Table 5 for the details which contains the length of L_{TReq} and L_{TRes}). The objective was to assess the number of discarded time responses TRes(D) received at the consumer node after successful time synchronization. These discarded responses, although no longer contributing to the synchronization, still require validation and checking the criteria to be processed, thereby introducing unnecessary overhead in communication and computation, particularly in real-time or resource-constrained environments.

We compared two configurations:

- Non-Systematic Configuration, where maximum values of m and n are fixed without accounting for channel conditions.
- Systematic Configuration,where m and n are dynamically set based on estimated parameters such as Bit Error Rate (BER), payload length, and required reliability, as derived from Algorithm 1.

The experimental results are presented in Table 7. They clearly demonstrate that the Systematic Configuration significantly reduces the number of discarded time responses while maintaining synchronization reliability. At 0% loss, the Non-Systematic Configuration results in 1860

Table 6
Impact of Systematic Configuration on safety application.

Parameter Symbol	Impact due to systematic configuration
m	Reduces redundant processing of SPDO, improves performance.
n	Reduces redundant processing of SPDO, improves performance.
TRC_k	Better in terms of performance and efficiency.
ts	Better in terms of ensuring safe operations.

discarded responses compared to only 42 for the Systematic Configuration, which is a reduction of approximately 97.7%. This observation continues across all levels of loss, with Systematic Configuration consistently reducing TRes(D) by a large margin. For instance, at 4% induced loss, Systematic Configuration leads to 350 discarded responses compared to 1761 in the Non-Systematic case, which is approximately 80.1% reduction.

It is worth noting that both configurations maintain successful synchronization across all tested loss levels, with a single synchronization failure occurring only at 4% loss in both cases. This outcome validates that the Systematic Configuration achieves equivalent reliability while offering superior efficiency. Furthermore, the Non-Systematic Configuration generates a high number of superfluous time responses regardless of the actual network conditions. Non-Systematic approach is quite inefficient, as it introduces fixed redundancy without considering the actual loss characteristics of the medium. On the other hand, the Systematic Configuration adapts the level of redundancy based on network conditions, which keeps a balance between reliability and resource utilization. These results support our proposed systematic method for configuring m and n, which leads to a more scalable, adaptive, and efficient time synchronization. It minimizes unnecessary pre-processing by reducing the volume of discarded TRes, thereby it improves the protocol's applicability in real-time systems and networks with constrained computational and communication resources.

6. Performance analysis test of openSAFETY protocol

In this section, we have performed the performance analysis of the openSAFETY protocol. Our experimental setup consists of three safety nodes within a dedicated Local Area Network (LAN) connected via Ethernet. These nodes are the SCM, the SN–Producer Node, and the SN–Consumer Node. The SCM is responsible for sharing the SNMT and SSDO frames, while the SN–Producer Node handles the transmission of Safety-Critical Data (SPDO) frames from the producer to the consumer node. The safety hardware, including the firmware SHNF, uses the UDP protocol. The experimental setup is illustrated in Fig. 12, and detailed information about the safety nodes is provided in Table 8.

We conducted two types of performance tests, as described below:

- Test Case 1: openSAFETY performance by varying the payload and transmission periods.
- Test Case 2: Network Impairment and Resilience Evaluation of openSAFETY Protocol.

6.1. Test case 1

In Test Case-1, we have shown the network conditions and the application requirements in Table 10. We evaluated the protocol's performance by varying the transmission period of SPDO frames and the payload size of the data. This test consisted of two experiments:

1. **Experiment 1:** We set the payload to half of the maximum capacity of a standard SPDO frame (See Table 11) and varied the transmission frequencies from 5 kHz to 100 Hz.

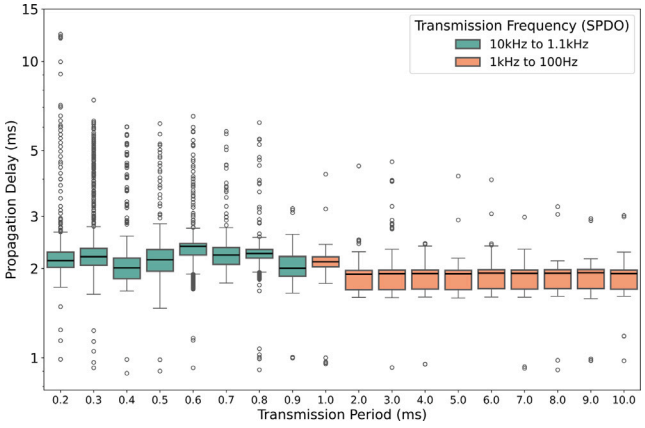

Fig. 15. Propagation Delay Experiment-1: Half Payload.

2. **Experiment 2:** We used the maximum payload capacity of a standard SPDO frame (See Table 11) and again varied the transmission frequencies from 5 kHz to 100 Hz.

A dedicated task was created using the Ptask library [24] to ensure the period update of the data in the SOD. Safety-critical data was exchanged for 60 s during each experiment, as outlined in Table 9. For each experiment, we have evaluated the performance based on the following metrics:

1. **SPDO Propagation delay (PD):** See Definition 22.
2. **Max Jitter** (J_{max}): The difference between Maximum and Minimum propagation delay is the maximum jitter during the transmission. Max Jitter can be represented as Max(PD)- Min(PD).
3. **Inter-Frame Delay Variation (IFDV):** The absolute value of the difference between the Propagation delay of two consecutive received SPDOs. IFDV can be expressed as $|PD(i+1) - PD(i)|$, where i is the order the frames were received [25].
4. **Bandwidth:** Bandwidth refers to the maximum rate at which data can be transferred over a network connection or communication channel Bandwidth = Frames/s * Payload.
5. **%Failure:** (SPDOs Failed/SPDOs Sent)*100.

6.2. Results test case 1

For both experiments, the propagation delay slightly decreases as the transmission period increases from 0.2 ms to 10 ms. The Box plots Fig. 15,16 highlight two frequency ranges: 5 kHz to 1.1 kHz (in green) and 1 kHz to 100 Hz (in orange). The 5 kHz to 1.1 kHz range, associated with shorter transmission periods, shows a slight increase in the propagation delay when the transmission frequency is high, but the mean propagation delay remains below 2.5 ms across all transmission frequency ranges.

Experiment 2 generally shows a slight variability compared to Experiment 1. In Experiment 1, the mean propagation delay lies between 1.8 and 2.4 ms. The number of outliers in the 5 kHz to 1.1 kHz range is higher than the 1 kHz to 100 Hz range. However, the distribution of propagation delay shows that most data points cluster around the mean values, indicating no significant variation in propagation delay for most cases.

In Experiment 2, the maximum jitter is slightly high at shorter transmission periods (up to 0.5 ms), reaching up to 20 ms. In contrast, Experiment 1 shows lower maximum jitter, reaching around 12 ms for a transmission period of 0.2 ms. We have shown the results of the experiments in Tables 12, 13. In Experiment 2 and Experiment 1, we observed that the mean IFDV is below 0.3 ms. The dots in

Table 7
Experiment results of Systematic and Non-Systematic Configuration of Number(s) of TReq(s) and TRes(s)

% Introduced loss	Configuration	No of m and n	TRes(D)	Sync-Fail
0%	Non-Systematic	m=63,n=63	1860	0
0%	Systematic	m=2,n=2	42	0
1%	Non-Systematic	m=63,n=63	1845	0
1%	Systematic	m=4,n=5	88	0
2%	Non-Systematic	m=63,n=63	1830	0
2%	Systematic	m=7,n=7	220	0
3%	Non-Systematic	m=63,n=63	1794	0
3%	Systematic	m=9,n=13	280	0
4%	Non-Systematic	m=63,n=63	1761	1
4%	Systematic	m=13,n=15	350	1

Table 8
Detailed information about safety nodes and Ethernet controller network.

Safety Node	CPU	Approx. Clock Drift	skew	OS and Kernel	Ethernet and Driver
SCM	Intel Core i7/3.9 GHz	12.212 ppm fast	1.338 ppm	Linux Mint 5.15.0–56	82541PI e1000
SN–Producer	Intel Core i5/2.9 GHz	20.612 ppm slow	0.255 ppm	Linux Mint 5.15.0–56	RTL8111 r8169
SN–Consumer	Intel Core i7/4.7 GHz	8.526 ppm slow	0.934 ppm	Linux Mint 5.15.0–56	RTL8111 r8169

Table 9
Baseline Test-Case 1.

Experiment ID	Payload	Transmission Frequencies
1	Half	5 kHz to 100 Hz
2	Full	5 kHz to 100 Hz

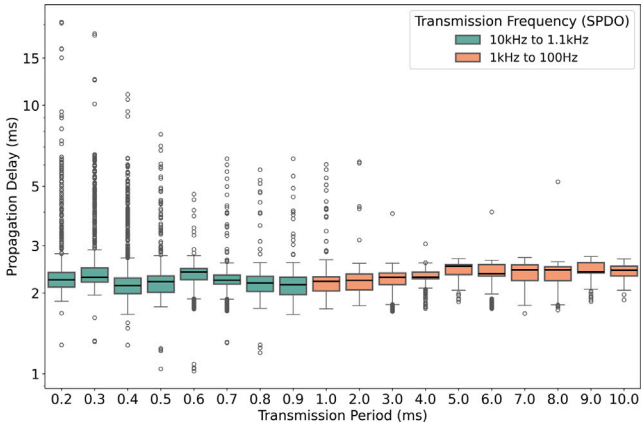

Fig. 16. Propagation Delay Experiment-2: Full Payload.

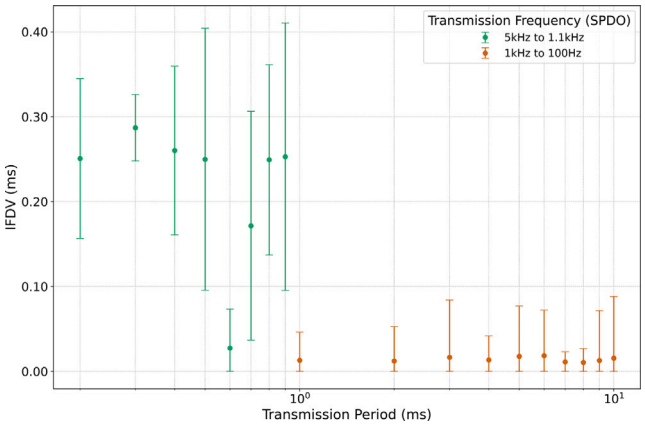

Fig. 17. IFDV Experiment-1: Half Payload.

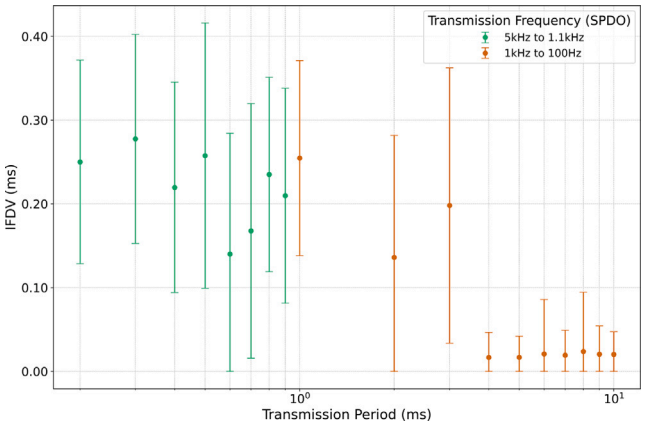

Fig. 18. IFDV Experiment-2: Full Payload.

the bar in Figs. 18, 17 represent the mean of IFDV, and the bars' length represents the standard deviation. At shorter transmission periods (0.2 ms to 0.9 ms), the mean IFDV usually remains between 0.20 ms and 0.30 ms; this indicates low variability in frame arrival times. However, at higher frequencies, the IFDV reaches up to 0.40 ms. This might be possible due to the increased data rate, which causes network congestion. Experiment 1 shows a lower mean IFDV across all transmission periods, showing improved stability when the Payload is half. The lower transmission frequency range (1 kHz to 100 Hz) consistently shows reduced IFDV in both experiments, indicating that lower transmission rates mitigate the impact of data variability. This disparity between the experiments shows the importance of optimizing transmission frequency and data load to minimize inter-frame delay variation. By leveraging lower transmission frequencies and adjusting the data load, the openSAFETY achieves more consistent frame intervals, which can help improve performance and stability.

In both experiments, Initially, at shorter transmission periods (0.2 ms to 0.5 ms), the bandwidth is higher due to the frequent transmission of frames. In Experiment 1, the peak bandwidth (consuming SPDO frames only) reaches approximately 12.5 Mbps; in Experiment 2, it reaches nearly 22.5 Mbps; the higher bandwidth observed in Experiment 2 shows increased network utilization under full Payload. The total message failures indicate that shorter transmission periods correlate with a higher incidence of failed messages. In both experiments, the number of failed messages increases when the transmission period reduces to 0.2 ms and 0.3 ms. Specifically, Experiment 2 shows

Table 10
Application design requirements: Test Case 1.

Parameter	P_e	CRERR	β_{cp}	SRT	RT	SNMT/SSDO timeout
Value	0.001	25%	0.2 ms	500 ms	10 ms	1 s

Table 11
Distribution of SPDO data Bytes under Half and Full Payload.

Payload	Data Bytes	Header-OS Bytes	Data-OS Bytes	Header-UDP Bytes	Header-IPV4 Bytes	Header-Ethernet Bytes	Total Payload Bytes
Half	128	13	2 × 128	8	20	14	311
Full	254	13	2 × 254	8	20	14	563

Table 12
Experiment-1 results.

Transmission period (ms)	Bandwidth (Mbps)	Mean PD (ms)	Max PD (ms)	Min PD (ms)	Max Jitter (ms)	Std Dev PD (ms)	SPDOs Sent	SPDOs Failed	Failure (%)
10.0	0.2488	1.8559	3.019	0.977	2.042	0.144	6001	0	0.0
9.0	0.2764	1.8696	2.946	0.977	1.969	0.1426	6667	0	0.0
8.0	0.311	1.8652	3.232	0.91	2.322	0.1413	7501	0	0.0
7.0	0.3554	1.8606	2.98	0.921	2.059	0.1408	8573	0	0.0
6.0	0.4147	1.8665	3.982	1.604	2.378	0.1431	10,001	0	0.0
5.0	0.4976	1.849	4.099	1.591	2.508	0.1445	12,001	0	0.0
4.0	0.622	1.8561	2.43	0.951	1.479	0.1413	15,001	0	0.0
3.0	0.8293	1.8524	4.579	0.927	3.652	0.1522	20,001	0	0.0
2.0	1.244	1.844	4.432	1.598	2.834	0.1439	30,001	0	0.0
1.0	2.488	2.1114	4.16	0.954	3.206	0.1061	59,997	0	0.0
0.9	2.7644	2.0372	3.186	0.998	2.188	0.1958	66,665	0	0.0
0.8	3.11	2.2395	6.221	0.911	5.31	0.1194	75,000	0	0.0
0.7	3.5543	2.2145	5.807	1.787	4.02	0.1967	85,710	0	0.0
0.6	4.1467	2.3243	6.535	0.925	5.61	0.1476	100,000	3	0.003
0.5	4.976	2.1419	6.174	0.902	5.272	0.2082	120,001	2	0.0017
0.4	6.22	2.0149	6.026	0.887	5.139	0.1758	150,001	18	0.012
0.3	8.2933	2.192	7.405	0.925	6.48	0.1742	200,001	66	0.033
0.2	12.44	2.1405	12.345	0.988	11.357	0.1958	300,001	377	0.1257

Table 13
Experiment-2 results.

Transmission Period (ms)	Bandwidth (Mbps)	Mean PD (ms)	Max PD (ms)	Min PD (ms)	Max Jitter (ms)	Std Dev PD (ms)	SPDOs Sent	SPDOs Failed	Failure (%)
10	0.4504	2.4253	2.679	1.887	0.792	0.1015	6001	0	0.0
9	0.5004	2.4642	2.738	1.858	0.88	0.1151	6668	0	0.0
8	0.563	2.3844	5.194	1.729	3.465	0.141	7500	0	0.0
7	0.6434	2.3969	2.711	1.678	1.033	0.1461	8572	0	0.0
6	0.7507	2.3852	4.006	1.749	2.257	0.1901	10,001	0	0.0
5	0.9008	2.4588	2.686	1.854	0.832	0.1159	12,001	0	0.0
4	1.126	2.3268	3.049	1.752	1.297	0.0948	15,001	0	0.0
3	1.5013	2.2392	3.95	1.707	2.243	0.1711	20,001	0	0.0
2	2.252	2.207	6.162	1.794	4.368	0.1814	30,001	0	0.0
1	4.504	2.1865	6.026	1.744	4.282	0.216	59,999	0	0.0
0.9	5.0044	2.1237	6.324	1.662	4.662	0.2062	66,666	0	0.0
0.8	5.63	2.1714	5.76	1.198	4.562	0.1787	75,000	0	0.0
0.7	6.4343	2.2512	6.326	1.305	5.021	0.1434	85,715	2	0.0023
0.6	7.5067	2.3359	4.668	1.024	3.644	0.1621	100,000	3	0.003
0.5	9.008	2.1695	7.806	1.044	6.762	0.2319	120,001	10	0.0083
0.4	11.26	2.1378	10.991	1.278	9.713	0.2152	150,000	95	0.0633
0.3	15.0133	2.334	18.509	1.317	17.192	0.1854	200,001	90	0.045
0.2	22.52	2.2474	20.44	1.28	19.16	0.1928	300,237	475	0.1582

a sharp spike in failed messages, exceeding 475 failures at 0.2 ms, compared to a lower count of around 377 failures in Experiment 1. This disparity highlights the increased likelihood of frame loss under higher data transfer rates and full payload conditions. As the transmission period goes ahead 0.7 ms, both experiments show a marked decline in message failures. This stabilization suggests that increasing the interval between transmissions allows the network to manage traffic more effectively, reducing frame loss and improving reliability.

6.3. Test case 2

Test Case 2 aims to test how well the protocol performs under adverse or non-ideal network conditions. We introduced controlled network impairments to evaluate the protocol's performance, including frame loss, latency, jitter, and bandwidth limitations using NETEM [22] in Linux. Ethernet networks are highly reliable, often exhibiting frame loss rates below 0.1%. However, testing up to 1% loss is reasonable to account for rare, extreme conditions according to IEEE 802.3 Ethernet Standard [11]. Ethernet LANs generally experience very low latency, typically between 1–5 ms, which is Common for Ethernet LANs with multiple switches or under moderate load. However, to simulate the overloaded or congested LAN environment, 10 ms latency can be used [26]. Small amounts of jitter (1–5 ms) can occur due to network congestion or switch buffering [27]. Ethernet speeds have evolved, ranging from 10 Mbps in legacy equipment to 1 Gbps in modern LANs. Testing across this range ensures the protocol can handle

Table 14
Baseline for Test Case 2.

Experiment ID	Frame Loss (%)	Latency (ms)	Jitter (ms)	Bandwidth (Mbps)	Description
1	0%	0 ms	0 ms	1000 Mbps	Normal
2	0.01%	0 ms	0 ms	1000 Mbps	Minimal frame loss
3	0.1%	0 ms	0 ms	1000 Mbps	Typical Ethernet LAN loss
4	1%	0 ms	0 ms	1000 Mbps	High frame loss
5	0%	1 ms	0 ms	1000 Mbps	Slight increase in latency
6	0%	5 ms	0 ms	1000 Mbps	Moderate latency increase
7	0%	10 ms	0 ms	1000 Mbps	High latency
8	0%	0 ms	1 ms	1000 Mbps	Low jitter
9	0%	0 ms	2 ms	1000 Mbps	Moderate jitter
10	0%	0 ms	5 ms	1000 Mbps	High jitter
11	0%	0 ms	0 ms	100 Mbps	Limited bandwidth
12	0%	0 ms	0 ms	10 Mbps	Severe bandwidth constraint
13	0.1%	5 ms	2 ms	1000 Mbps	Moderate loss, latency, and jitter
14	1%	10 ms	5 ms	10 Mbps	Severe loss, latency, and low bandwidth

Table 15
Results for Test Case 2: Under degraded network conditions.

Experiment ID	Min PD (ms)	Mean PD (ms)	Max PD (ms)	Std Dev PD (ms)	Mean IFDV (ms)	Max Failed SPDOs	Failure (%)
1	1.732	2.2384	9.997	0.2201	0.2198	0	0.0
2	1.647	2.1474	14.106	0.2809	0.2205	1	0.0017
3	1.835	2.2677	17.992	0.2666	0.2184	13	0.0217
4	1.655	2.2626	8.128	0.2351	0.2185	23	0.0383
5	2.714	3.1176	18.987	0.2491	0.221	0	0.0
6	6.769	7.2248	15.763	0.2099	0.2228	0	0.0
7	11.752	12.1304	17.791	0.1881	0.2218	0	0.0
8	1.872	2.4816	13.912	0.3384	0.3328	2591	4.3183
9	1.696	2.3175	18.560	0.3857	0.3141	6053	10.0883
10	1.696	2.3507	18.267	0.3880	0.2696	9553	15.9217
11	1.717	2.1777	17.458	0.2623	0.221	1	0.0017
12	39.500	58.5641	79.400	10.5432	0.1888	720	1.2000
13	1.600	5.0232	15.000	1.3362	0.1856	1750	2.9100
14	7.000	57.1092	77.800	12.3871	1.2852	26,124	43.5400

both constrained and high-speed scenarios. The IEEE 802.3u [28] and 802.3ab [29] standards define Fast Ethernet (100 Mbps) and Gigabit Ethernet (1000 Mbps), respectively [11].

Experimental setup test case 2

The experimental setup for Test Case 2 is precisely the same as for Test Case 1. Additionally, we have fixed the transmission frequency to 1 KHz and the Payload to the Full Payload for this test. The baseline of introduced network impairments to perform these experiments are shown in Table 14.

6.4. Results test case 2

The experimental evaluation of the communication protocol under degraded network conditions shows its robustness and handling of network impairments. The results of the experiment are shown in Table 15.

In the baseline test (Experiment 1), the protocol shows stable performance with a low mean latency of 2.24 ms, minimal variation, and no frame loss, establishing an ideal reference point.

As frame loss is introduced incrementally in Experiments 2–4, there is a slight increase in mean latency (from 2.15 ms to 2.27 ms) and standard deviation, indicating a minor impact on delay. The failure rate remains very low, reaching 0.0383% in Experiment 4, showcasing the protocol's resilience under typical Ethernet LAN conditions. In Experiments 5–7, increasing latency without additional faults increases mean latency (up to 12.13 ms) as expected. Notably, the protocol maintains a zero failure rate, even as the network delay increases; this highlights the ability to tolerate added propagation delay without sacrificing reliability. The introduction of jitter in Experiments 8–10 causes noticeable increases in delay variation and the number of failed messages. The reason is that the jitter can cause frames to arrive out of order. This can disrupt protocols expecting frames to arrive in sequence, potentially affecting the data integrity or requiring reordering mechanisms; therefore, it causes the failure rate to rise significantly, reaching up to 15.92% in Experiment 10. Despite this, the mean latency remains relatively stable, suggesting that the protocol can handle moderate jitter but struggles when the variability becomes severe. Bandwidth limitations in Experiments 11 and 12 substantially increase mean latency, particularly in Experiment 12 (mean PD of 58.56 ms). Although the protocol performs well under moderate constraints (Experiment 11), severe bandwidth limitations result in a failure rate of 1.2%, indicating congestion and potential frame loss under restricted bandwidth conditions.

Combined fault conditions in Experiments 13 and 14 illustrate the compounded effects of multiple network impairments. In Experiment 13, the protocol manages moderate faults with a mean latency of 5.02 ms and a failure rate of 2.91%. However, Experiment 14, involving severe frame loss, latency, jitter, and bandwidth constraints, pushes the protocol to its limits, with the highest failure rate at 43.54%. This indicates a significant degradation in performance, as expected under extreme network conditions.

7. Conclusions

OpenSAFETY offers significant flexibility and adaptability as a fieldbus-independent protocol, making it an ideal choice for industrial safety systems. Our research into the time synchronization mechanism has provided critical insights into maintaining the operational integrity of safety nodes, particularly in real-time systems where deterministic timing is essential. A significant contribution of this work is the development of a structured approach for configuring safety applications. To the best of our knowledge, no existing method offers practical

guidelines for configuring safety nodes, and our approach addresses this gap by providing a clear, effective, and structured method for tuning safety applications to ensure reliable and safe operations. This method and the associated algorithm can be deployed in real industrial systems to tune the openSAFETY protocol and related application parameters based on real-time network conditions and application requirements.

In industrial systems, where the communication network may experience varying levels of congestion, packet loss, or delays, the proposed parameter tuning will be beneficial in maintaining safe and efficient openSAFETY operations. This approach also represents a step toward isolating application-dependent and protocol-dependent parameters. Additionally, when used as a real-time monitoring defensive technique within the Safety Node (SN), the SCM can trigger the parameter adjustment process in safety nodes. This process will recalculate the time synchronization parameters, raising an alarm when the current configuration parameters are no longer suitable or autonomously adjusting the time synchronization parameters to ensure they remain within a range verified and approved by the industry's safety team. For example, when a new safety node is added to the network or an existing node is removed, the algorithm can adjust the time synchronization parameters autonomously. This process simplifies operations for end-users who lack protocol-level knowledge.

The parameter tuning approach and the real-time parameter monitoring algorithm can be integrated into an easy-to-use software interface for deployment in real industrial systems. This software interface would enable engineers and system designers to set high-level objectives, such as the desired reaction time or required redundancy for time synchronization, while the monitoring algorithm handles the lower-level adjustments based on current network conditions and application needs.

The software interface would also allow monitoring of the system's performance, including parameters such as latency, jitter, and packet loss, and enable the safe and controlled adjustment of key parameters if necessary, based on ongoing analysis of network performance and safety requirements. This is especially important in wireless networks [30], where the performance is sometimes unpredictable due to interference, multipath fading, or signal attenuation. In such systems, the algorithm can modify some key parameters like the Time Request Cycle (TRC), Time Synchronization Timeout (ts), and Time Delay (Td) to compensate for the loss and delay in transmission.

One of the important features of the tuning algorithm is its ability to adapt to network impairments. If the system detects that the communication medium is experiencing high levels of packet loss or high jitter, the algorithm can increase the redundancy of time requests and responses. This can increase the possibility that the safety nodes continue to synchronize properly even under suboptimal conditions. However, further work is needed to comprehensively analyze the protocol's performance in wireless systems, as wireless networks introduce more unpredictable delays and variable network conditions, making it challenging to predict best and worst-case delays.

Future studies should focus on detailed evaluations of how openSAFETY can be optimized further for wireless communication, considering factors such as signal interference, multipath propagation, and real-time packet loss recovery policies. Additionally, the adaptation of the protocol under highly fluctuating network conditions, such as those seen in industrial wireless systems, will be important for future deployments.

As a part of this work, we have examined the impact of non-systematic parameter configurations and demonstrated how our approach is helpful. The systematic effects of parameter configurations on safety applications are summarized in Table 6. Furthermore, we conducted a comprehensive performance analysis of openSAFETY, evaluating its handling of safety-critical data transmission over UDP via Ethernet. Our results revealed that the protocol successfully maintains its expected cycle time across varying payload sizes. We also assessed its performance under different network impairments and found that openSAFETY remains robust and reliable, even in network disruptions under a certain level.

CRediT authorship contribution statement

Shoaib Zafar: Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Conceptualization. **Salvatore Sabina:** Writing – review & editing, Visualization, Validation, Supervision, Investigation. **Alessandro Biondi:** Validation, Supervision, Resources, Project administration, Investigation. **Giorgio Buttazzo:** Validation, Supervision, Resources, Project administration, Investigation, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We are thankful to B&R automation for providing the openSAFETY stack for research purpose.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.sysarc.2025.103605.

Data availability

The authors do not have permission to share data.

References

[1] G. Peserico, A. Morato, F. Tramarin, S. Vitturi, Functional safety networks and protocols in the industrial internet of things era, Sensors 21 (18) (2021) 6073, http://dx.doi.org/10.3390/s21186073.

[2] HMS Networks, Annual analysis reveals steady growth in industrial network market, 2024, URL: https://www.hms-networks.com. (Accessed 24 November 2024).

[3] G. Creech, Black channel communication: What is it and how does it work? Meas. Control. 40 (10) (2007) 304–309, http://dx.doi.org/10.1177/002029400704001003.

[4] B.I. Automation, Wireless transmission of safety data, 2018, https://www.br-automation.com/en/about-us/press-room/wireless-transmission-of-safety-data-24-09-2018/. (Accessed 24 November 2024).

[5] I.E. Commission, IEC 61508 Commented version, 2.0, 2020, IEC, 2010, Stability date.

[6] F. Haslhofer, OpenSafety for wired and wireless connections over MQTT, 2020, URL: https://epub.jku.at/download/pdf/5335415.pdf. (Accessed 24 November 2024).

[7] A. Hadziaganović, M.K. Atiq, T. Blazek, H.-P. Bernhard, A. Springer, The performance of opensafety protocol via IEEE 802.11 wireless communication, in: 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, 2021, pp. 1–8, http://dx.doi.org/10.1109/ETFA45728.2021.9613548.

[8] A. Hadžiaganović, R. Muzaffar, H.P. Bernhard, A. Springer, Integration of opensafety in omnet++, in: IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society, 2022, pp. 1–6, http://dx.doi.org/10.1109/IECON49645.2022.9968888.

[9] A. Soury, M. Charfi, D. Genon-Catalot, J.M. Thiriet, Performance analysis of ethernet powerlink protocol: Application to a new lift system generation, 2015, pp. 1–6, http://dx.doi.org/10.1109/ETFA.2015.7301492.

[10] E.T. Group, Ethercat technology, 2024, https://www.ethercat.org/en/technology.html. (Accessed 24 November 2024).

[11] IEEE Computer Society, IEEE Standard for Ethernet, Technical Report IEEE Std 802.3-2018, IEEE Standards Association, New York, NY, USA, 2018, http://dx.doi.org/10.1109/IEEESTD.2018.8457469, Revision of IEEE Std 802.3-2015.

[12] PROFIBUS, P. International, PROFIBUS technology, 2024, https://www.profibus.com/technologies/profibus. (Accessed 24 November 2024).

[13] International Organization for Standardization (ISO), Safety of machinery — Safety-related parts of control systems. Part 1: General principles for design, Technical Report ISO 13849-1:2023, ISO, 2023, (Accessed 24 November 2024).

[14] G. Cena, I.C. Bertolotti, S. Scanzio, A. Valenzano, C. Zunino, Synchronize your watches: Part II: Special-purpose solutions for distributed real-time control, IEEE Ind. Electron. Mag. 7 (2) (2013) 27–39, http://dx.doi.org/10.1109/MIE.2013.2248431.
[15] T. Krauss, R. Schierl, Safety-critical systems based on PROFIsafe: An overview, in: Proceedings of the International Conference on Industrial Networks and Intelligent Systems, 2016, pp. 15–22.
[16] H. Networks, Canopen protocol whitepaper, 2024, https://media.hms-networks.com/image/upload/v1701953901/Documents/Whitepapers/Ixxat_CANopen-Protocol-Whitepaper_EN.pdf. (Accessed 24 November 2024).
[17] G. Peserico, A. John, Development of functional safety applications for autec products. Study of protocols: Canopen, canopen safety, FSOE, and ProfiSafe, 2019/2020, https://hdl.handle.net/20.500.12608/23978. (Accessed 24 November 2024).
[18] D.D. Chowdhury, NextGen Network Synchronization, Springer International Publishing, 2021, http://dx.doi.org/10.1007/978-3-030-71179-5, (Accessed 24 November 2024).
[19] J. Peeck, M. Möstl, T. Ishigooka, R. Ernst, A protocol for reliable real-time wireless communication of large data samples, IEEE Trans. Veh. Technol. 72 (10) (2023) 13146–13161, http://dx.doi.org/10.1109/TVT.2023.3275300.
[20] A. Platschek, B. Thiemann, H. Zeilinger, T. Sauter, An error model for safe industrial communication, in: IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society, 2015, pp. 004672–004677, http://dx.doi.org/10.1109/IECON.2015.7392829.
[21] C. KS, 61000-4-4: Electromagnetic compatibility (EMC)-part 4: Testing and measurement techniques-section 4: Electrical fast transient/burst immunity test, Basic EMC Publ. Korean Stand. Assoc. (2009).
[22] Linux Foundation, Tc-netem(8) - linux manual page, 2024, URL: https://www.linux.org/docs/man8/tc-netem.html. (Accessed 18 November 2024).
[23] C. Project, Chrony: A versatile implementation of the network time protocol (NTP), 2024, https://chrony-project.org/index.html. (Accessed 30 April 2024).
[24] G. Buttazzo, G. Lipari, Ptask: An educational c library for programming real-time systems on linux, in: Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation, ETFA, IEEE, 2013, pp. 1–8, http://dx.doi.org/10.1109/ETFA.2013.6648001.
[25] J.H. Salim, J.M. Halpern, E. Shenker, Terminology for traffic control mechanisms, 2006, http://dx.doi.org/10.17487/RFC4689, RFC 4689 URL: https://www.rfc-editor.org/rfc/rfc4689.
[26] G. Almes, S. Kalidindi, M.J. Zekauskas, A one-way packet loss metric for IP performance metrics (IPPM), 1999, http://dx.doi.org/10.17487/RFC2680, RFC 2680. URL: https://www.rfc-editor.org/rfc/rfc2680.
[27] A. Morton, B. Holbrook, H. Schulzrinne, Packet delay variation applicability statement, 2009, http://dx.doi.org/10.17487/RFC5481, RFC 5481. URL: https://www.rfc-editor.org/rfc/rfc5481.
[28] IEEE Computer Society, IEEE standard for information technology – telecommunications and information exchange between systems – local and metropolitan area networks – specific requirements – part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications – amendment 3: Media access control (MAC) parameters, physical layer, medium attachment units, and repeater for 100 mb/s operation (100base-t), Technical Report IEEE Std 802.3u-1995, IEEE Standards Association, New York, NY, USA, 1995.
[29] IEEE Computer Society, IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific Requirements – Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications – Amendment 5: Physical Layer Specifications for 1000 Mb/s Operation on Four-Pair Category 5 Cabling (1000BASE-T), Technical Report IEEE Std 802.3ab-1999, IEEE Standards Association, New York, NY, USA, 1999.
[30] Schildknecht, Wireless opensafety via bluetooth, 2025, URL: https://www.schildknecht.ag/en/products/industrial-wireless/wireless-opensafety/. (Accessed 24 August 2025.

Shoaib Zafar is currently pursuing a Ph.D. in Emerging Digital Technologies at Scuola Superiore Sant'Anna, Pisa, Italy. He is affiliated with the Real-Time Systems (ReTiS) Laboratory, where his research focuses on hard real-time communication protocols and safety mechanisms in embedded systems. He is being supervised by Prof. Alessandro Biondi and Prof. Giorgio Buttazzo. He obtained his Master of Engineering in Computer Science and Technology from Harbin Institute of Technology, China, in 2021, under the supervision of Prof. Weizhe Zhang. He has also worked as a Visiting Researcher at the University of Western Macedonia, Greece. His research interests include functional safety, industrial communication protocols, cyber–physical systems, and the safety and security of embedded systems.

Salvatore Sabina received the M.S. degree in electronic engineering from the University of Pisa in 1983. From 1985 to 1998, he involved in satellite applications, hardware designs, and HRT operating systems. In 1999, he joined Ansaldo Segnalamento Ferroviario (now Hitachi Rail STS) as responsible for the Product Development Department. At the end of 2022, he completed his professional career in Hitachi Rail STS as Vice President of the STS Innovation Department. He is currently acting as S.Anna's Professional Affiliate (TeCIP) for setting up innovative projects and collaborations with research and development organizations.

Alessandro Biondi is associate professor at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna.He graduated (cum laude) in Computer Engineering at the University of Pisa, Italy, within the excellence program, and received a Ph.D. in computer engineering at the Scuola Superiore Sant'Anna under the supervision of Prof. Giorgio Buttazzo and Prof. Marco Di Natale. In 2016, he has been visiting scholar at the Max Planck Institute for Software Systems (Germany). His research interests include design and implementation of real-time operating systems and hypervisors, schedulability analysis, cyber–physical systems, synchronization protocols, and safe and secure machine learning. He was recipient of six Best Paper Awards, one Outstanding Paper Award, the ACM SIGBED Early Career Award 2019, the IEEE TCCPS Early Career Award 2023, and the EDAA Dissertation Award 2017.

Giorgio Buttazzo is full professor of computer engineering at the Scuola Superiore Sant'Anna of Pisa. He graduated in Electronic Engineering at the University of Pisa, received a M.S. degree in Computer Science at the University of Pennsylvania, and a Ph.D. in Computer Engineering at the Scuola Superiore Sant'Anna of Pisa. He has been Editor-in-Chief of Real-Time Systems, Associate Editor of IEEE Transactions on Industrial Informatics and ACM Transactions on Cyber-Physical Systems. He is IEEE fellow since 2012, wrote 7 books on real-time systems and more than 300 papers in the field of real-time systems, robotics, and neural networks, receiving 15 best paper awards.