Simulation framework for EtherCAT over TSN

Balakrishna Balakrishna
NGNI
Fraunhofer FOKUS
Berlin, Germany
https://orcid.org/0000-0002-7168-8565

Boris Meinardus, Leonidas Kontopoulos
NGN / AV
Technical University of Berlin
Berlin, Germany
{b.meinardus, kontopoulos}@campus.tu-berlin.de

***Abstract*—EtherCAT is a real-time industrial Ethernet technology suitable for hard and soft real-time requirements. It is based on new technology, "on-the-fly processing", which significantly reduces cycle times and jitter to further expand the capabilities of IIoT. IEEE 802.1 TSN in recent years gained more attention in IIoT because it offers real-time communication over a shared Ethernet medium with predictable end-to-end delay and jitter. However, there is no work presented to showcase the benefits of EtherCAT and TSN integration apart from theoretical information. Therefore, to expedite the analysis of EtherCAT over a TSN network, we used OMNeT++. Here, we will compare the cycle time and jitter by evaluating different scenarios of EtherCAT communications over a NeSTiNg TSN network.**

***Index Terms*—TSN, EtherCAT, Network Simulation, Real-Time Communication, IIoT, Industry 4.0.**

I. Introduction

INDUSTRY 4.0 is the next step in combining traditional manufacturing methods with the upcoming highest-end technologies. Cyber-Physical Systems (CPS) are the main focus in these technologies and they are well-structured systems comprised of different entities, which directly interact with the physical world while being controlled, operated, and integrated by a centralized core. The core could analyze the data received by sensors and execute the tasks for the system.

Usually, the CPS and its entities are connected over a network and the network has unpredictable jitter and unforeseeable delay. It is necessary to minimize these characteristics while working with time-critical data. To meet the hard real-time requirements in time-sensitive CPS, deterministic communication is obligatory.

In order to sufficiently minimize jitter and delay, many different Fieldbus technologies are developed such as EtherCAT, ProfiNet, etc. Each Fieldbus technology is developed for a specific application's requirement and has its advantage over the other Fieldbus technology. Development of new Fieldbus technology is always necessary due to new requirements in the latest industrial applications. The introduction of new Fieldbus technology does not only bring advantages to deterministic communication but also adds disadvantages such as new hardware type, incompatibility with legacy Fieldbus technologies, etc. Therefore, Time-Sensitive Networking (TSN) is evolved to address these disadvantages to provide deterministic communication in an Industry.

ISBN 978-3-903176-39-3

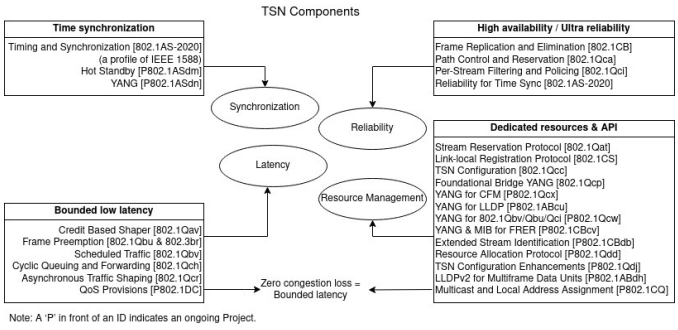

Fig. 1. List of IEEE 802.1 TSN standards [1]

IEEE created the IEEE 802.1 TSN Task Group (TG) to develop TSN and the TG formally known as Audio Video Bridging (AVB) TG. TSN is a set of standards specified by IEEE 802.1 as shown in Fig. 1, which enable the transportation of different traffic class over a network through the use of Ethernet. High transportation speed and simplification of the network structure are two of the most significant benefits offered. TSN operates in Layer 2 of Open Systems Interconnection (OSI) model and it does not have an application profile. Therefore, enabling legacy Fieldbus Technologies to TSN opens many advantages such as resource management, ultra-reliability, bounded low latency and time synchronization.

Many Field-level devices manufacture provide TSN specification [2] [3] [4] for their new devices but the existing Filed-level devices are still incompatible with TSN. Moreover, only a few automation networks over TSN [5] [6] [7] are analysed. Clock synchronization between EtherCAT and TSN is only assessed in [5]. Therefore, it is necessary to investigate the differences using EtherCAT with and without TSN. Since the availability of TSN compatible legacy Field-level devices in the market are few, this paper focuses on EtherCAT over TSN in a network simulation tool to analyse cycle time and jitter.

EtherCAT short for Ethernet for Control Automation Technology, developed by the EtherCAT Technology Group is first introduced back in 2003 as a real-time industrial Ethernet technology. EtherCAT is suitable for hard and soft real-time requirements, by using Ethernet as a transport medium for a Fieldbus. Its main goals are guaranteeing a reduced cycle time to less than 100 μs and jitter to less than 1 μs [8]. Their approach to achieving those goals is by some new innovative

features, one of which is on-the-fly processing.

TSN components *IEEE 802.1 Qbv*, also known as Time Aware Shaper (TAS), and *IEEE 802.1 AS-Rev*, also known as profile of Generalized Precision Time Protocol (gPTP) are the minimum two standards required for deterministic communication in a TSN domain. In short, TAS propagates the incoming data to the respective egress port. Here, the data will be buffered in a queue representing its traffic class before a Transmission Selection Algorithm determines which data will be selected first. The gPTP is responsible for the time-synchronization of all nodes in the network with a common and aligned schedule for deterministic event handling. Therefore these two TSN sub-standards are most relevant for this project.

Multiple alternatives exist to analyze the effectiveness and proper functionality of EtherCAT over the TSN network. Real-time application is one of them but the lack of necessary equipment and resources can render it to be quite the struggle. Mathematical analysis is another. However, to analyze bigger and more complex networks it is necessary to describe the network topology mathematically, which is not trivial. From previous experience obtained in [9], we decided to go with the more subtle approach: simulation. By utilizing Objective Modular Network Testbed in C++ (OMNeT++) and two open-source projects, Network Simulator for TSN (NeSTiNg) [10] which covers the IEEE 802.1 TSN network and EtherCAT_Simulation [11], we managed to create a depiction of intended network to inspect.

OMNeT++ is an open-source Integrated Development Environment (IDE), which uses C++ as its main programming language and NEtwork Description (NED) as its network description language. By combining the two languages it is possible to create any kind of wireless or wired communication simulation for testing and experimentation. This will help us test out our frameworks in a controlled environment.

In order to proceed, we first need to integrate the frameworks to run our simulations and extract data. One of our tasks is to check the compatibility of the two frameworks and make the required changes to effectively combine the two domains. After the successful combination of the frameworks, we need to evaluate performance characteristics and create proof-of-concept test cases by comparing the best and worst-case communication over a TSN network.

We will begin with a brief description of the concept of EtherCAT and TSN in Section II, followed by Section III provides the necessary information of the two frameworks (NeSTiNg and EtherCAT) before diving deeper into our approach. Finally, we will analyze the performance simulations in Section IV and conclude with our findings and possible future work at Section V.

II. Background and Related Work

A brief overview of EtherCAT and TSN is described in this section. In addition to this, a short literature survey is provided. This survey focuses on field-level communication integration into TSN.

A. EtherCAT Overview

As briefly mentioned in the introduction, EtherCAT is a real-time industrial Ethernet technology suitable for real-time communication requirements. A typical EtherCAT network consists of one master and a finite number of slaves. To increase the predictability of the network, the master is the only one allowed to generate packets. The most common topology encountered is the ring and daisy chain topology, while a star topology is also possible. In the case of a daisy chain setup, the master sends out a frame that is transmitted from one slave to the next. Each slave processes the data addressed to it without making an exact copy of the entire frame, while at the same time continuing the propagation of the frame. The frame is read on the physical layer, therefore only adding a hardware propagation delay and manifesting a predictable network performance. This concept is considered "on-the-fly processing" and is EtherCAT's main feature.

How does each slave acknowledge which datagram is addressed to it, if the slave is not making a copy of the ingressing frame? That is where EtherCATs unique frame structure becomes relevant. The frame that is being circulated through the network is of Ethernet structure since EtherCAT is an Ethernet-based framework. The EtherCAT functionalities are found in the payload, where they consist of a 2 Byte long header and the datagrams destined for the slaves. The EtherCAT header consists of 11 bits for the length of all datagrams combined, a reserved bit and 4 bits for the protocol type. The header is followed by the datagrams destined for the slaves as showcased in Fig.2. Each datagram has a 10 Byte long header, data, and a 2 Byte long Working Counter (WKC).

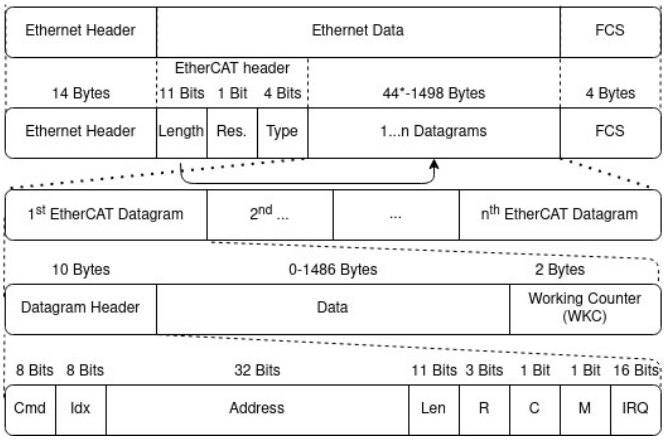

Fig. 2. EtherCAT frame structure

As shown in Fig. 2, the first 8 bits contain the command which the slave shall execute. A list of those commands can be found in [12]. EtherCAT master uses four possible addressing types to communicate between and slave that are Positional, Node, Logical and Broadcast. The master uses positional addressing to locate all the slaves. Node addressing is usually used when the master wants to access registers of individual and discovered slaves. Broadcasting, as the word

itself indicates, addresses all slaves in the segment to check the status of each slave or initialize them. Each slave has a Fieldbus Memory Management Unit (FMMU), which it uses to translate logical addresses to physical addresses and store the data in its local addressing space.

B. TSN Overview

TSN implements multiple scheduling mechanisms, where *IEEE 802.1 Qbv* being the most important one. It is responsible for splitting the communication on the medium into fixed cycles, which in some ways is similar to Time Division Multiple Access (TDMA). The fixed cycles, also referenced as time slices, are issued based on Virtual Local Area Network (VLAN) priority. By creating VLAN communication channels, more time-critical data can easily be separated from Best-Effort (BE) data, thus conflicts between the two of them can be avoided.

Each TSN capable node has TAS implemented in its ports. There are up to eight queues, which are controlled by gating mechanisms to determine when buffered packets will be propagated to their next destination. Each packet is equipped with a VLAN tag, which contains a three-bit Priority Code Point (PCP). Based on the PCP value, the packet is enqueued in the corresponding queue. For an enqueued packet to be forwarded over a port, the gate responsible for the respective queue needs to be opened. The order under which gates open is determined by a Gate Control List (GCL). Here, a gating schedule defines the duration of the entered gate states. After the last gate state is terminated, the schedule is reset. Multiple ports of different devices can have codependent GCL. In order to keep those synchronized and predictable, *IEEE 802.1 AS-Rev* is used.

IEEE 802.1 AS-Rev, is a profile of gPTP, enables the highest precision of time synchronization. The core principle is the usage of a Grandmaster clock and hardware produced timestamps. Frame residence is calculated in each bridge and the propagation delay between two neighboring switches. The Grandmaster clock is the root of all timing after which selected masters coordinate themselves and thus the slaves in their segments.

C. Related Work

The clock synchronization between EtherCAT and TSN is evaluated in [5] using UPPAAL model checker. The authors develop three UPPAAL models with and without TSN network inside the EtherCAT network domain. After evaluating the three models, the author concluded that the proposed clock synchronization mechanism for the EtherCAT-TSN network achieves at least 3 times higher clock synchronization precision compared to not using any clock synchronization. Although the authors have provided the basic insight on the EtherCAT-TSN clock synchronization, still the shared Ethernet medium feature is not evaluated using *IEEE 802.1 Qbv*.

In [6], a simulation approach is introduced for field-level communication over TSN, which is Sercos over TSN. Sercos Master/Controller is connected to its devices using a TSN switch in tree topology with three short lines. The TSN switch also consists of BE talkers to exhibit the shared medium feature. The authors demonstrated the TSN TAS feature to protect the time-critical Sercos Frames from interfering with BE traffic. In a test case where the tree topology is consists of 13 Sercos slave devices and a jitter $\pm 0.9\mu s$ is identified where the duration is $1.3s$, the cycle time is $2ms$ and the obtained values are 650. Here, TAS feature of TSN helped to achieve the jitter ($\leq 1\mu s$). After evaluating multiple use cases with different combination of data rate and topology, the authors have concluded that Sercor over TSN brings more benefits such as huge bandwidth and topology migration.

Apart from the work presented above, only specification [2] [3] [4] is available for TSN integration by other field-level device manufacturers. To analyze EtherCAT performance over TSN, we considered [4] as a reference architecture in this paper.

III. Approach In Connecting Domains

A. EtherCAT Framework

The main difficulty while constructing an EtherCAT framework is implementing a structure that can replicate the on-the-fly processing feature. We have chosen to use [11] as our EtherCAT framework, which implements a Type 12 Process Data Units (PDU) system to realize the feature. The framework sends out each byte after a provided time interval(byte-to-byte delay). This way the EtherCAT slave is processing one byte and propagates it before the next one arrives at its gate. Besides the byte-to-byte delay, a frame-to-frame delay is present, which is responsible for the timing at which the succeeding frame will be scheduled. For every single byte created, the following function is used to determine its scheduled time:

$$scheduletime = d_b \cdot n_b + d_f \cdot n_f \tag{1}$$

d_b: Delay byte to byte, n_b: Index of current byte to send,
d_f:Delay frame to frame, n_f: Index of current frame sending

The payload size of each frame and the number of EtherCAT frames that will be sent out are predetermined before the initialization of the simulation. Once the frame is sent, it is propagated throughout a daisy chain, and this is the only implemented topology. However, certain EtherCAT specific characteristics, such as WKC, Interrupt Request (IRQ), RESERVED, C-bit, NEXT, and DATA, are implemented but never used, and Distributed Clocks are also absent. Therefore, direct communication between master and slaves is a basic level implementation of the physical layer.

B. NeSTiNg Framework

NeSTiNg framework [10] developed by the University of Stuttgart and managed to establish a very accurate and precise version of the real-world application [13], by extending the INET [14] library. It implements multiple modules, such as different hosts and TSN capable switches to create a coherent and synchronized network. Out of the IEEE 802.1 standards required to create a TSN network, the two most relevant to us, *IEEE 802.1 Qbv* and *IEEE 802.1 AS-Rev*, are applied in

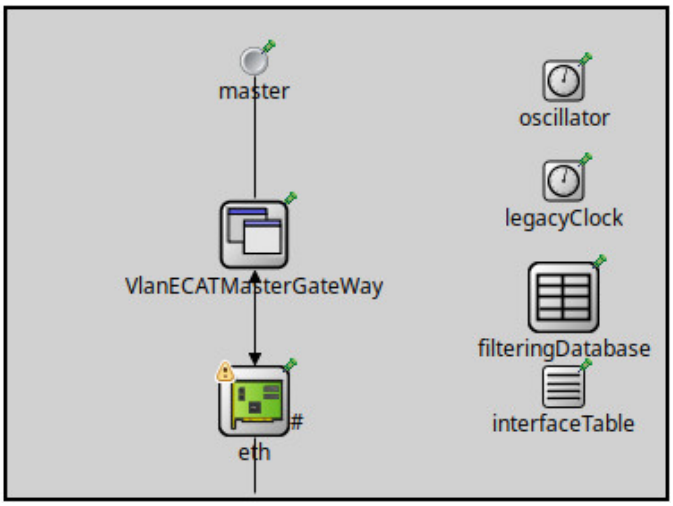

Fig. 3. OMNeT++ NeSTiNg_ECAT_Master Module. Here, the 3 submodules used for the NeSTiNg_ECAT_Master Module can be observed. The manipulated master submodule, VlanECATMasterGateWay and Ethernet interface card

an interesting manner. *IEEE 802.1 Qbv* is most noticeably integrated into the ports of the TSN switches. Here, besides the usual encapsulation and decapsulation process of the frames, 8 queues are implemented to replicate the PCP values of *IEEE 802.1p* [15]. Each one of these queues is controlled by a gating mechanism. Which gates will be open at what time is determined by a bit vector and the provided duration of that configuration. When one of the gates is open, the data enqueued in the respective queue is propagated to a Transmission Control Algorithm (TCA), which determines the order in which the packets will be sent out, based on their priority.

On the other hand, *IEEE 802.1 AS-Rev* conspicuously differs from the real-world application. While TSN integrates the gPTP to synchronize its network, NeSTiNg does by having a global discrete clock that every device in the network is connected to it. This way, every device is indirectly connected and synchronized with one another, leading to a representative application of the gPTP. Each time a packet is received by the switches and processed, the opening of the egress gates is determined by the clock. However, the main difference between gPTP and the usage of a global clock is that the gPTP is more susceptible to unsynchronized behaviour in case the grandmaster clock fails. In this case, a new election needs to be started, during which all devices in the network may not be synchronized. With a global clock, such failures are avoidable. Therefore, a global discrete clock is used for *IEEE 802.1 AS-Rev* substitution.

C. EtherCAT and NeSTiNg Framework Integration

To implement a TSN capable master module, we have decided to use the already existing basic EtherCAT master from the EtherCAT framework as a submodule for our NeSTiNg_ECAT_Master as shown in Fig. 3. In the first step, we had to adjust the byte-to-byte delay. To be more precise, it is set to zero since it is not suitable for a standard frame transmission, thus having the master send out all bytes of a frame at the same time. In order to propagate the frame through a NeSTiNg TSN Network, the frame needs to be extended with tags such as a VlanID, a Drop Eligible (DE) bit, the PCP value and all the necessary Ethernet header characteristics. For this purpose, we need to develop a gateway that can extend or rather convert the frame sent out by the basic EtherCAT master into the NeSTiNg suitable frame structure. To create the gateway, we copied the VlanTraffGenScheduler module from NeSTiNg and made a few modifications to it. Once the gateway receives all the bytes of one frame, it first modifies the Ethernet header and includes a VLAN tag. For the payload of the packet, we used INETs Byte Chunk [16] structure to copy all the necessary information related to the EtherCAT datagrams.

Since all the bytes received from the EtherCAT Master are empty, we only focused on inserting the Type 12 PDUs in the payload. To make the Master TSN compatible, an Ethernet Interface needs to be implemented that enables *IEEE 802.1 Qbv*. Once the payload is created together with the Ethernet header, it is packed into an INET packet before forwarded to the Ethernet Interface. Here, the packet is encapsulated with the missing Frame Check Sequence (FCS) and gets enqueued in its corresponding queue of the egress port. The packet's PCP value is set to be the highest in the network since it is time-critical. Based on the GCL that is defined in an Extensible Markup Language (XML) data file, together with the GCL of every device in the Network, the respective gate awaits its opening to propagate the packet in the network. Every port, where the EtherCAT packet is propagated through, has the same schedule as the EtherCAT Master module to synchronize the network.

The NeSTiNg_ECAT_Slave module is very similar to the NeSTiNg_ECAT_Master, with the only key difference being the substitution of the EtherCAT Master with the Slave. Once the packet arrives at the Slave Gateway, it gets decapsulated and sent to the EtherCAT slave one byte after the other. There a small processing delay is added and the bytes return to the gateway ready to be sent back to the master. This process restarts for every frame.

IV. Evaluation

This project aims at simulating a network, in which time-triggered and non-time-triggered traffic is transmitted to determine how beneficial the integration of a TSN network inside an EtherCAT infrastructure would be. Next, we will showcase the main networks we tested to achieve our final results.

A. Worst case network

In Fig. 4, our main network setup is showcased with one time-triggered device (MACMaster), and two non-time-triggered workstations (*Workstation 1* and *Workstation 2*). All three devices, combined with the MACSlave and backupServer, share the same link between *SwitchA* and *SwitchB*. All links have a bandwidth of $1 Gbits^{-1}$ and a propagation delay of $20ns$. Each queue at every port is capable of storing

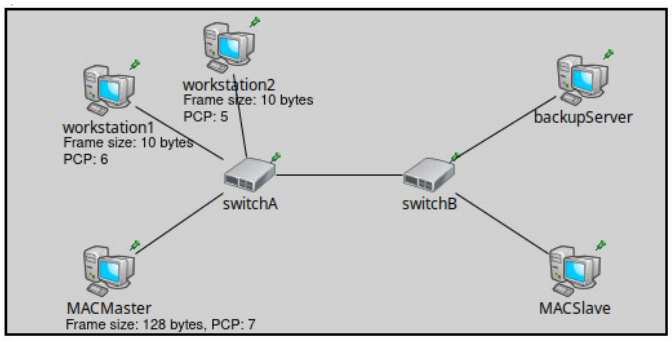

Fig. 4. Exemplary worst-case network test case with three traffic flows, one of which is the time-critical EtherCAT communication

30 MTU-sized frames (MTU of 1522 bytes). Workstation 1 and Workstation 2 are BE hosts and they are assigned with the PCP values 5 and 6 respectively, while MACMaster receives the value 7 for the highest priority. The workstations send out 10 bytes sized frames, while the frames generated by the master have a payload of 100 bytes, which accumulates to 128 bytes in total for the Ethernet packet. Both time-critical and BE data are given the same cycle time of $3\mu s$. Since we want to minimize any additional queuing delay, the GCL has the same cycle time as the MACMaster, i.e every $3\mu s$ the gate for queue 7 opens to propagate the EtherCAT frame. For the transmission of a 128 bytes frame on a $1Gbits^{-1}$ line the delay is $1.072\mu s$, hence the gate for queue 7 is open for $2\mu s$ and the remaining gates are open for $1\mu s$. To draw conclusions, we used the same setup as showcased in Fig. 4 and changed certain parameters to create different test cases as seen in Table I. We also created a best test case, where the additional traffic is removed together with the GCL and the traffic between the MACMaster and MACSlave is constant. To evaluate the implementation of EtherCAT over TSN we compared the Round Trip Time (RTT) and the jitter.

Name	Description
Test Case 1	TSN functionalities
Test Case 2	TSN functionalities with an additional slave
Test Case 3	All traffic streams assigned with different PCP (no TAS)
Test Case 4	All traffic streams assigned with the same PCP (no TAS)
Test Case 5	Best test case (only EtherCAT traffic stream presents)

TABLE I
DIFFERENT TEST CASES

B. Performance Evaluation

After presenting the basic functionality of our implementation, we continue with the evaluation of our simulations.

1) Setup: All simulations are performed on a machine with 4 Intel I7-7700HQ processors at a clock speed of 2.8 GHz, and a total RAM of 16 GB, running on Ubuntu version 18.04.4 LTS. The used OMNeT++ simulation tool version is 5.5.1.

2) Parameters: For our worst test case (Test Case 1), we scheduled our MACMaster to send out frames size of 128 bytes on every $3\mu s$. In this case, TAS is enabled and the EtherCAT frames are assigned with the highest PCP value. Test Case 2 is the same setup, with one extra slave. Therefore, the payload size expands to 218 bytes. Test Case 3 uses again one slave but the TAS is removed by constantly having all the gates open. In test case 4, TAS is absent and the PCP values of the Workstations are the same as the MACMaster, which fully removes the main TSN functionalities. Finally, Test Case 5 is the best test case, where all additional traffic is removed and the gates of the switches are constantly open. Each simulation is run 10 times for $100\mu s$, with the number of frames ranging from 100 to 1000 with steps size 100. For the final results, the simulations with 1000 frames are used.

3) Results: Fig. 5 depicts the calculated RTT for each test case. The theoretical RTT can be calculated through $d_{tot} = 2 \cdot (2 \cdot d_{proc} + 3 \cdot (d_{trans} + d_{prop}))$, where d_{proc} is the processing delay of the switches, which is assigned random values between $300ns$ and $500ns$ to further add an element of realism, d_{prop} the propagation delay and d_{trans} the transmission delay. For test case 1 we calculate the total RTT is $\approx 8.312\mu s$. The value is just half from the one shown in Fig. 5 because of the additional queuing delay. When the frame arrives at the switches, it first needs to be processed. Due to the change of the gating state from the GCL while frame processing, the window under which it can be sent out is almost closed, leading to its gate getting shut for $1\mu s$. When this additional queuing delay is added to the RTT, we strive for a value closer to $\approx 16.312\mu s$, which closely resembles the one showcased. Since the gates are strictly controlled, the RTT of the first two test cases leads to constant. This manifests the correct implementation of the TAS creating a deterministic and predictable traffic flow. On the other hand, test case 3, 4, and 5 have much smaller RTT. This is because of the absence of a GCL, hence adding no extra queuing delay leads the test case to a RTT that is closer to the one calculated at $\approx 8.312\mu s$. However, the graphs only showcase the traffic of the EtherCAT frames. If the gates are constantly open, the additional traffic gets heavily delayed, which leads to further delays when more time-critical infrastructures are connected to the network.

Fig. 6 shows the jitter of each test case and the jitter is measure on RTT after reception of EtherCAT frame inside the EtherCAT master. Since the RTT of the first two cases is constant, the accumulated jitter is zero, further proving the desired predictability of the TSN network. The remaining cases have strongly differing values for the first 100 frames received, but this phenomenon is due to the random processing delay. Once enough frames have been received, they all start equaling out into a constant line at approximately $1.7\mu s$.

4) Conclusion: In conclusion, the demonstrated performance evaluation shows that the integration of TSN in an EtherCAT network comes with additional processing and queuing delays due to the addition of gating mechanisms. However, it renders the entire network synchronous and predictable, hence keeping the time-critical element that is obligatory for EtherCAT. Combined with further domains, such as ProfiNET, SERCOS III, the overall implementation cost can be decreased by sharing a medium with slightly increasing the total delay.

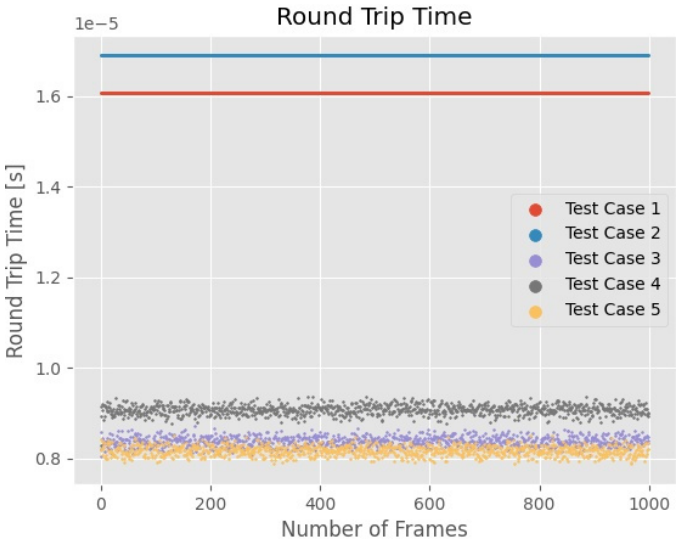

Fig. 5. Round Trip Time of Frames: The time it takes for the EtherCAT frame to propagate to the EtherCAT slave and back to the EtherCAT master

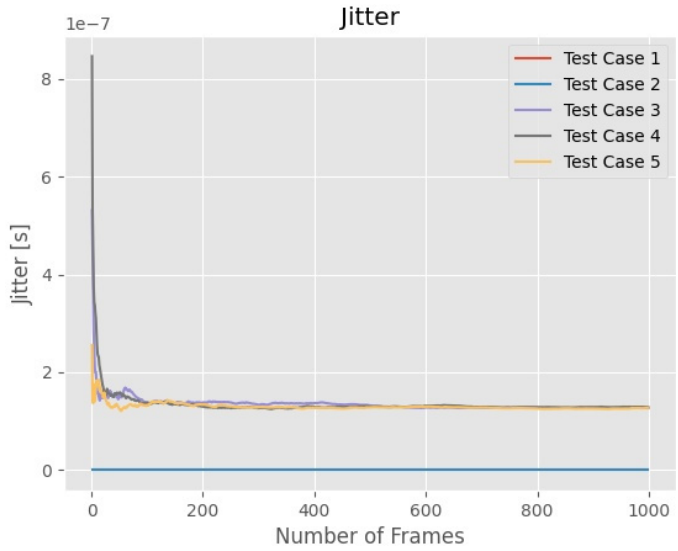

Fig. 6. Jitter updated on each frame reception in the EtherCAT master.

V. Summary And Future Work

In this paper, we noticed the implementation of EtherCAT over TSN is possible. Additional hardware required that properly encapsulates and decapsulates the packets and it is synchronised with the TSN Network. One of the main goals of EtherCAT is to have a RTT that is smaller than $100\mu s$ and jitter of less than $1\mu s$. From our performance evaluations, we can showcase that it is possible to accomplish these numbers since our RTT is constant at approximately $16\mu s$ and the jitter is approximate to zero. Both these values do not perfectly represent a real-world application but come very close to it.

Nevertheless, our simulation is very minimal because there are certain factors NeSTiNg does not support, for example, *IEEE 802.1 AS-Rev*. Additionally, the sending intervals between frames is constant and the EtherCAT functionality is not fully implemented in the EtherCAT framework. Despite these little imperfections we managed to set the foundation for the simulated integration of EtherCAT over TSN.

For future work, a separate TSN gateway can be introduced for EtherCAT devices because existing EtherCAT devices do not support TSN functionalities and it is essential to analyze their performance with TSN integration. Furthermore, changes to the EtherCAT framework can be made to improve its similarity to the real-world application. For example, the daisy chain topology of the slaves can be improved by adjusting the propagation and transmission delay between slaves, thus possibly needing to adjust the cycle time. Additionally, the EtherCAT frame format used can be adjusted to accurately resemble the real-world application, by adding a detailed Ethernet header and datagrams. *IEEE 802.1 AS-Rev* can also be introduced to verify the time synchronization between EtherCAT and TSN Network because the time synchronization of EtherCAT and TSN devices provides more insights on RTT and jitter.

References

[1] Time-Sensitive Networking (TSN) Task Group. [Online]. Available: https://1.ieee802.org/tsn/

[2] PROFINET over TSN Guideline. [Online]. Available: https://www.profibus.com/download/profinet-over-tsn

[3] CAN to TSN. [Online]. Available: https://www.cast-inc.com/interfaces/automotive-bus-controllers/can2tsn/

[4] EtherCAT and TSN. [Online]. Available: https://www.ethercat.org/en/ethercat_and_tsn.htm

[5] D. Mateu, D. Hallmans, M. Ashjaei, A. Papadopoulos, J. Proenza, and T. Nolte, "Clock Synchronization in Integrated TSN-EtherCAT Networks," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, October 2020, pp. 214–221.

[6] S. Nsaibi, L. Leurs, and H. D. Schotten, "Formal and Simulation-based Timing Analysis of Industrial-Ethernet Sercos III over TSN," in *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, October 2017.

[7] P. Pfrommer, A. Ebner, S. Ravikumar, and B. Karunakaran, "Open Source OPC UA PubSub over TSN for Realtime Industrial Communication," in *Emerging Technologies in Factory Automation (ETFA)*, July 2018, pp. 1087–1090.

[8] EtherCAT - the Ethernet Fieldbus. [Online]. Available: https://www.ethercat.org/en/technology.html

[9] M. Pahlevan, B. Balakrishna, and R. Obermaisser, "Simulation Framework for Clock Synchronization in Time Sensitive Networking," in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, July 2019, pp. 213–220.

[10] NeSTiNg Omnet++ Simulation Framework. [Online]. Available: https://gitlab.com/ipvs/nesting

[11] EtherCAT Omnet++ Simulation Framework. [Online]. Available: https://github.com/ayoubsoury/ethercat-simulation

[12] Ethercat Hardware Data Sheet Section I. [Online]. Available: https://download.beckhoff.com/download/document/io/ethercat-development-products/ethercat_esc_datasheet_sec1_technology_2i2.pdf

[13] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel, "NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++," in *2019 International Conference on Networked Systems (NetSys)*, March 2019.

[14] INET Omnet++ Simulation Framework. [Online]. Available: https://github.com/inet-framework/inet

[15] PCP value definitions. [Online]. Available: https://en.wikipedia.org/wiki/IEEE_P802.1p

[16] Working with Packets. [Online]. Available: https://inet.omnetpp.org/docs/developers-guide/ch-packets.html