An EDF-based Swapping Approach to Enhance Support for Asynchronous Real-Time Traffic over EtherCAT networks

Gaetano Patti, Lucia Lo Bello, Giuliana Alderisi, Orazio Mirabella
Department of Electrical, Electronic and Computer Engineering
University of Catania
Catania, Italy
{gaetano.patti, lucia.lobello, giuliana.alderisi, orazio.mirabella}@dieei.unict.it

Abstract

EtherCAT is a real-time Ethernet protocol for factory automation applications that enables periodic data exchange with cycle times of a few microseconds thanks to the capability to process frames "on-the-fly". EtherCAT does not provide efficient mechanisms for enabling the transmission of asynchronous real-time data from slave nodes. This paper proposes an EDF-based Swapping Approach that allows slaves to send real-time asynchronous traffic over EtherCAT networks in an efficient way, while maintaining the interoperability with standard devices.

The paper describes the EDF-based Swapping Approach and presents a comparative performance evaluation with both the EtherCAT standard and an approach in the literature that exploits CAN-Like arbitration.

1. Introduction

EtherCAT is a Real-Time Ethernet (RTE) protocol suitable for factory automation applications. It is included as part of the IEC 61158 standard [1][2], which defines fieldbus protocols for industrial communications, and in the IEC 61784 [3] standard, which defines the RTE Communication Profiles. EtherCAT provides a daisy-chain topology and a master/slave architecture in which the master periodically transmits standard Ethernet frames containing multiple telegrams (called Type 12 PDUs in the IEC 61158 standard). Slaves read and/or write data in the telegrams by processing the frame "on-the-fly". This solution provides cycle times lower than one millisecond, e.g., a 1500-byte EtherCAT frame is processed in about 150 us. The EtherCAT protocol allows the transmission of both real-time and non real-time traffic. The periodic traffic is assumed to be real-time, while asynchronous (i.e., aperiodic) traffic may have or not have real-time constraints. Asynchronous real-time traffic is scheduled in the same way as periodic traffic. The master cyclically sends one or more EtherCAT frames which contain the telegrams. On the contrary, asynchronous non real-time traffic is handled by means of special frames, called *mailboxes*. The master sends those frames to the slaves that have previously signaled the need for asynchronous transmission. However, the transmission of asynchronous real-time traffic is not efficient, as it needs to be scheduled using at least one telegram for each slave that required transmission, with the result of increasing the cycle time.

To allow slaves to autonomously transmit asynchronous traffic, in the literature an approach is proposed that uses a CAN-Like arbitration scheme to transmit asynchronous messages in a contended EtherCAT telegram [10]. The approach requires the master to send an acknowledgement on message reception, which allows the slave to remove the successfully transmitted message from its local queue. In addition, the approach in [10] assumes that the message priority is static.

An improvement of [10] is proposed in the work in [11], which foresees that multiple messages per telegram can be transmitted, however a master acknowledgment message is still needed.

This paper proposes a novel approach that allows the transmission of asynchronous traffic with real-time constraints over EtherCAT, while maintaining the interoperability with standard EtherCAT devices. The approach, here called an EDF-based Swapping approach, enables the slaves to swap an incoming asynchronous message based on an Earliest Deadline First algorithm [15], so that messages with closer absolute deadlines preempt those with farther ones. No asynchronous message will be lost due to preemption from other messages, as the slave that has swapped the incoming message will be in charge of transmitting it whenever possible according to the EDF rule.

The goal of the proposed approach is to allow the slaves transmitting asynchronous real-time data without significantly increasing the cycle time of periodic real-time traffic. Moreover, the EDF-based Swapping approach provides messages with dynamic priorities and, thanks to the swapping mechanism, eliminates the need for the slaves to wait for the master acknowledgement

message, thus improving the asynchronous real-time throughput.

The paper is organized as follows. Section 2 outlines related works. Section 3 summarizes the EtherCAT protocol. Section 4 introduces the EDF-based Swapping approach, while Section 5 presents a comparative performance assessment through OMNeT++ simulations. Finally, Section 6 concludes the paper and outlines future work.

2. Related Works

Many studies addressed EtherCAT with the aim of assessing its performance and proposing improvements. The work in [4] presents a performance comparison between EtherCAT and Powerlink in the context of a coordinated motion control application. In [5] and [6] a comparison between EtherCAT and PROFINET IRT is presented, while in [7] the EtherCAT performance with different topologies is investigated. In [8] a switch that significantly reduces propagation delays is proposed.

EtherCAT provides a clock synchronization protocol called Distributed Clocks (DC). The synchronization accuracy of this protocol, evaluated in [9], is always better than 1 microsecond.

A limitation of EtherCAT is the limited support provided to asynchronous traffic, as the protocol was specifically designed for periodic traffic.

In order to overcome such a limitation, in [10] a CAN-based arbitrating transmission scheme is proposed, that allows EtherCAT slaves to transmit asynchronous real-time data without the need for scheduling those messages as periodic traffic. In the approach, a new EtherCAT telegram, containing an asynchronous message with an assigned priority, is defined. A slave with a message ready for transmission can "on-the-fly" replace the content of the telegram with its own, if the ready message to be transmitted has a higher priority than the incoming message in the telegram. The process continues until the asynchronous telegram reaches the master, which will then notify all the slaves about the content of the received telegram.

The approach in [10] therefore enables asynchronous messages transmission without the need for the master to poll the slaves and avoids scheduling time-constrained asynchronous messages as periodic data, as this would entail a bandwidth waste. However, the approach proposed in [10] has some drawbacks. For instance, as message priorities are static, under high asynchronous workloads low priority messages, due to interference from high priority ones, would experience long delays, with a potential for starvation for the lowest priority messages. Another drawback is the need for an acknowledgement mechanism to notify the slaves about the message that has been successfully received, so as to allow the slave that transmitted the message to remove it from its local queue. The acknowledgment is realized by the master sending an "ack" telegram that contains a copy of the latest received message. The need for the slave to wait for the "ack" prevents the possibility to embed multiple asynchronous telegrams in an EtherCAT frame. This problem was overcome by the approach proposed by the same authors in [11], which augments the approach in [10] with the capability of embedding multiple asynchronous messages within a single telegram. However, the need for acknowledgment still remains, thus reducing the bandwidth efficiency.

The idea proposed in this paper is inspired by the Slot Swapping Protocol (SSP) that was presented in [12], [13], and [14] to interconnect heterogeneous fieldbus networks through a real-time backbone. In the SSP protocol, fieldbus gateways are connected according to a ring topology. The ring is covered uninterruptedly by a sequence of slots, where each slot carries a message. A node connected to the backbone maintains a local queue of messages ordered according to their absolute deadline (which depends both on arrival time and on the temporal validity of the message). A node can swap an incoming message with its own if, and only if, the deadline of the packet in the local queue is closer than that of the message that is traversing the node. If this is the case, the node replaces the ongoing message with its own, and stores the swapped message in its local queue.

The introduction of an SSP-based approach in EtherCAT enables dynamic priority for asynchronous messages, thus overcoming the limitations found in [10] and [11]. The swapping mechanism also allows to immediately remove a message from the slave queue and also provides the capability of embedding multiple asynchronous telegrams in one EtherCAT frame.

3. EtherCAT Protocol features

EtherCAT [1][2] has a master/slave architecture in which the master cyclically sends an Ethernet frame to slaves according to a daisy-chain topology. Slaves read and/or insert data into PDUs called Type 12 PDUs, which are processed "on the fly", i.e., the slaves process one byte at a time and forward it to the next node. When the last slave of the chain is reached, the frame is redirected backward to the master as a response frame and the cycle ends. This behavior requires that all nodes be full-duplex devices capable of receiving and transmitting at the same time.

The Ethernet frame contains the Type 12 frame, which encapsulates one or several Type 12 PDUs. Fig. 1 shows the EtherCAT frame structure. The Type 12 frame supports three types of data:

- Type 12 PDUs, for transmitting process data.
- Network variables, for network management purposes.
- Mailboxes, for transmitting standard IP packets and non real-time data.

The Type 12 PDU includes a header, which specifies the address and command for a slave, (i.e., read, write or

read/write), a data field and a Working Counter Field (WCF) used for error detection.

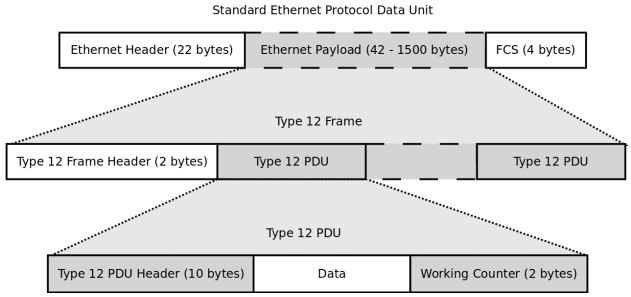

Fig. 1. EtherCAT frame structure with telegrams.

An important index to evaluate the EtherCAT performance is the minimum cycle time (T_C) [4][5], i.e., the minimum time needed to exchange the input/output data between the master and all the slaves [5], defined as in formula (1) [4]:

$$T_C = T_{et} + T_{ec} + T_{de} + T_{if} \qquad (1)$$

where

- T_{et} is the time necessary to transmit the Ethernet header and Frame Check Sequence (FCS) fields.
- T_{if} is the interframe gap, i.e., the time between the end of frame and the beginning of the next one.
- T_{de} is the frame delay, i.e., the time introduced by each slave for processing the frame. Assuming that such a delay is the same for all the slaves, $T_{de}=N\times T_{sv}$, where N is the number of slaves and T_{sv} is the slave processing time.
- T_{ec} is the time necessary to transmit the Type 12 frame, specified by formula (2) [4]:

$$T_{ec} = T_{eh} + L \times (T_{th} + T_{wc}) + \sum_{i=1}^{L} T_{ct}^{(i)} \qquad (2)$$

where T_{eh} is the time necessary to transmit the Type 12 frame header, L is the number of Type 12 PDUs in a Type12 frame, T_{th} and T_{wc} represent the transmission time of the Type 12 PDU header and working counter, respectively. Finally, T_{ct} is the time to transmit the i-th Type 12 PDU payload.

Formulas (1) and (2) show that the minimum cycle time directly depends on both the number of Type 12 PDUs and the length of their payloads. Moreover, the maximum payload size of the Type 12 frame is 1500 bytes and a large number of Type 12 PDUs entails the need to use more Ethernet frames, with a consequent increase of the minimum cycle time.

4. The EDF-based Swapping Approach

The EtherCAT protocol does not allow one or more slaves to transmit messages in an autonomous way. The transmission of non real-time traffic is realized through special Type 12 frames, called mailboxes, but it is up to the master polling the slaves so as to enable them to transmit mailboxes.

To allow the slaves to transmit asynchronous traffic, many solutions can be adopted. For instance, one solution is to periodically poll the slaves, in order to determine which slaves have asynchronous data to transmit. This is accomplished through a single Fieldbus Memory Management Unit (FMMU), i.e., an entity that allows a mapping between the slave physical memory and a portion of a Type 12 PDU, thus providing slaves with the possibility of reading/writing data in a single telegram. After collecting the information about the slaves that have asynchronous data to transmit, the master schedules a mailbox frame for each of them, thus significantly increasing the cycle time. Mailboxes are not suitable for transmitting real-time asynchronous messages, as there is no way to provide guarantee about the message delivery time.

Another solution is to handle asynchronous traffic as periodic traffic using the Type 12 PDU. This option may be suitable for asynchronous real-time traffic because a Type 12 PDU is reserved for each slave, thus guaranteeing transmission. However, this solution has a drawback, as it reserves a telegram to the potential transmission of asynchronous data without any guarantee that there will actually be data to transmit. If no data are available for transmission, this approach entails an unnecessary increase of the cycle time and bandwidth waste.

The new idea proposed in this paper is to introduce a novel Type 12 PDU, called Async Type 12 PDU, to be used by slaves for sending asynchronous traffic when they really need to transmit. The Async Type 12 PDU is contented between all the slaves that have asynchronous real-time data to send. Contention is handled according

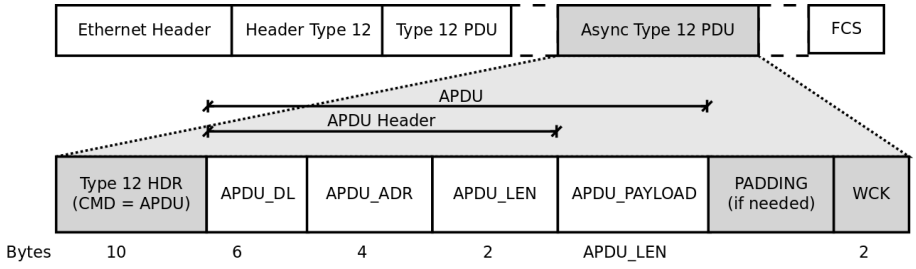

Fig. 2. EtherCAT Frame structure with Async Type 12 PDU

to a preemptive policy based on the Earliest Deadline First (EDF) algorithm [15]. This is possible, as the daisy-chain topology used in EtherCAT allows for preempting messages by changing "on-the-fly" the telegram payload of an incoming frame when it goes through a slave. The number of Async Type 12 PDUs to be embedded in a Type 12 frame is set during the configuration phase, depending on both the constraint on the cycle time and the asynchronous real-time workload generated and its constraints, also taking into account the maximum Ethernet frame payload (1500 bytes).

The frame structure is shown in Fig. 2. The Async Type 12 PDU contains an Asynchronous PDU (APDU) which is composed of a header, a payload and, if required, a padding. The Async Type 12 PDU has fixed length and its payload is the APDU. The APDU header fields are the ones listed below, i.e.,

- APDU_DL (6 bytes): the APDU deadline, which is used to take the swapping decision.
- APDU_ADR (4 bytes): the address of the slave that has sent the latest message received by the master.
- APDU_LEN (2 bytes): the length of the APDU payload.

In order to maintain the coexistence between the slaves that implement the EDF-based Swapping Approach and those that implement the EtherCAT standard, the Async Type 12 PDU header is mapped on the standard Type 12 Header as specified in Table 1.

The CMD field provides a new value (e.g., 0x0F) that indicates that the Type 12 PDU must be processed as asynchronous PDU. The ADR field contains the address of the slave that has sent the latest message received by the master.

Data Field	*Data Type*	*Value/description*
CMD	Unsigned8	Command: APDU (0x0F)
IDX	Unsigned8	Index
ADR	DWORD	Slave address of last message
LEN	Unsigned11	Length of DATA field
RESERVED	Unsigned3	0x00
C	Unsigned1	Circulating Frame
NEXT	Unsigned1	0 if the last PDU in the frame
IRQ	WORD	Reserved for future use
DATA	OctetString	Data
WKC	WORD	Working Counter

Table 1. Async Type 12 PDU Fields

Each APDU has an associated absolute deadline that depends on the message expiring time.

A slave generating an APDU calculates the absolute deadline by adding the relative deadline of the message received by the Application layer to the system time. The System Time variable is 64 bits long and contains the nanoseconds elapsed since January 1, 2000 [9]. As the approach here proposed does not require nanosecond accuracy, the absolute deadline is expressed in microseconds.

Due to serial communication, to perform deadline comparison on-the-fly, the six bytes of the deadlines are encoded from the most significant byte to the least significant byte, and their transmission follows the same order.

The modules implementing the EDF-based Swapping Protocol are shown in Fig. 3.

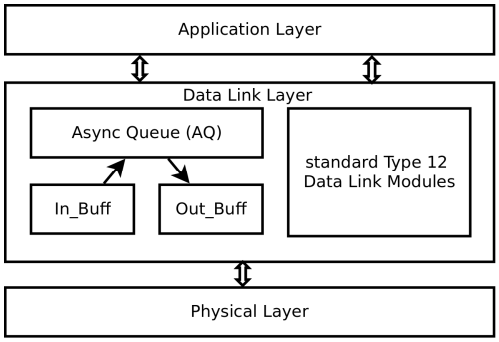

Fig. 3. Data Link Layer modules for the EDF-based Swapping approach

Each slave maintains a local queue (Asynchronous Queue, AQ) of APDUs ordered according to their absolute deadlines. This is possible, as the EtherCAT protocol specifies a clock synchronization protocol called Distributed Clocks (DC)[9][1], that provides a synchronization accuracy in the order of nanoseconds.

When a slave needs to send an APDU (i.e., a local message, M_{lo}, stored in the output buffer Out_Buff) and is being traversed by an Async Type 12 PDU containing another APDU (i.e., an incoming message, M_{in}, stored in the input buffer *In_Buff*), it can swap the incoming APDU with the local one according to the EDF rule, i.e., if, and only if, M_{lo} has a closer deadline than M_{in}. In this case, the slave inserts the swapped APDU in its local queue according to its deadline.

The comparison works as follows. The i-th byte of the deadline of the incoming message M_{in} (i.e., the APDU_DL field of the APDU), $B_{i,in}$, for i=0...5, is compared with the corresponding byte of the APDU_DL field of the APDU of the local message M_{lo}, $B_{i,lo}$. The incoming message M_{in} is swapped if, and only if, the following inequality (3) is true

$$B_{i,in} > B_{i,lo} \quad (3)$$

otherwise M_{in} is forwarded to the next slave.

If a swap occurs, while the local message M_{lo} is transmitted to the next slave and removed from the local queue, the swapped message has to be entirely received and is then inserted in the local queue according to its deadline.

The first advantage of this approach is that no asynchronous message will be lost due to preemption

Data field	*Symbol*
Number of slaves	N
Periodic payload size	S_{PP}
Number of Async Type 12 PDUs	N_{APDU}
APDU size	S_{APDU}
Slave latency	t_{sv}
Application message relative deadline	DL_{APP}
Application start time	APP_{START}
Async Mean Generation Interval	λ
CAN-Like message priority	CL_{PRIO}

Table 2. Notation used for simulation parameters

from other messages, as the slave that has swapped the incoming message will be in charge of transmitting it whenever possible according to the EDF rule.

Moreover, the EDF-based transmission policy provides messages with dynamic priority, thus preventing the starvation problems that affect lower priority messages under fixed priority scheduling.

5. Simulations

In order to evaluate the EDF-based Swapping approach, a simulator has been developed using the OMNeT++ framework [16]. The goal of the simulations is to assess the performance of the EDF-based Swapping approach and to make a performance comparison with the EtherCAT standard as far as the transmission of asynchronous real-time data from the slaves is concerned. Moreover, a comparison between the EDF-based Swapping approach and the CAN-Like approach proposed in [10] is also presented.

As shown in Fig 4, the evaluated scenarios consist of a single EtherCAT segment with 10 slaves. The master is directly connected to the first slave and periodically sends Ethernet frames. In all the simulations, the periodic real-time workload consists of one 32-byte telegram sent from each slave every cycle. In these simulations no mailboxes are used for transmitting real-time asynchronous traffic, as mailboxes are not able to support real-time constraints, so they are not suitable for real-time asynchronous traffic. Moreover, potential transmission errors are not taken into account, as they are beyond the scope of this paper. The notation used in the following part of this Section is shown in Table 2. Simulation time was 500 ms.

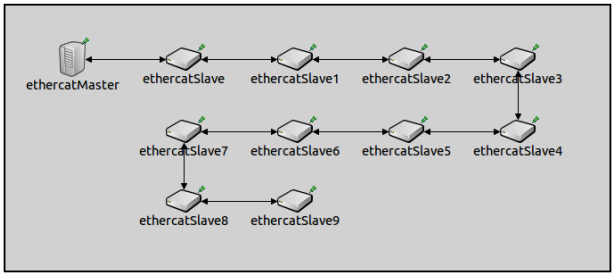

Fig. 4. Simulated network topology

5.1. Performance evaluation of the EDF-based Swapping Approach

In the first set of simulations, the performance of the EDF-based Swapping Approach is evaluated by varying the mean generation interval of asynchronous messages from all slaves. The asynchronous workload has been generated using an exponentially distributed random function with mean λ. Several simulations were run by varying the number of Async Type 12 PDUs embedded in each EtherCAT frame.

The performance parameter chosen for this scenario is the Deadline Miss Ratio (DMR), calculated at the Application layer of the master and expressed as the ratio between the number of asynchronous real-time messages that miss their deadline and the total number of asynchronous real-time messages generated. Note that in this paper we are not addressing or applying any admission control tests, so deadline misses for asynchronous messages may occur. Late asynchronous messages, i.e., those that exceed their deadlines, may be either dropped or not, depending on the application (firm or soft real-time). In our simulations we decided not to drop late asynchronous messages, so as to assess the network behavior under higher workloads.

5.1.1. Simulation 1: DMR assessment by varying the asynchronous network workload and the number of Async Type 12 PDUs.

This simulation evaluates the DMR obtained by varying the network asynchronous workload, while keeping a constant periodic workload. Simulation parameters are specified in Table 3.

Symbol	*Value/range*
N	10
S_{PP}	32 bytes
N_{APDU}	from 1 to 7
S_{APDU}	32 bytes (12 header + 20 payload)
t_{sv}	700 ns
DL_{APP}	500 us
APP_{START}	exponential (25 us)
λ	from 112.5 us to 1000 us

Table 3. Simulation 1 parameters

The increased number of Async Type 12 PDUs, on the one hand, provides more room for asynchronous traffic, on the other hand it increases the EtherCAT frame length and thus also the minimum cycle time.

The maximum DMR is the highest DMR value obtained before the network approaches saturation.

Table 4 shows the values of the maximum Deadline Miss Ratio for asynchronous messages as a function of the mean message generation period obtained by varying

the number of Async Type 12 PDUs embedded in one EtherCAT frame for each simulation run.

By increasing the number of Async Type 12 PDUs, higher workloads can be supported, thus reducing the maximum DMR values. However, as the second column in Table 4 shows, this is at the expense of longer cycle times. As a result, at design time the tradeoff between the maximum asynchronous workload that can be supported and the corresponding cycle time value has to be taken into account.

Number of Async Type 12 PDU	*Minimum Cycle Time (T_C)*	*Maximum DMR (%)*	*Corresponding λ*
1	50.68 us	3.7980	600 us
2	54.36 us	3.5769	300 us
3	58.04 us	3.5050	212.5 us
4	61.72 us	1.5087	175 us
5	65.40 us	1.5076	150 us
6	69.08 us	1.8290	125 us
7	72.76 us	1.9733	112.5 us

Table 4. Max. DMR for asynchronous messages as a function of the number of Async Type 12 PDUs with different mean generation intervals in a non-saturated network.

Fig. 5 shows the details of the simulation performed with N_{APDU} = 1 to show the trend of DMR as a function of the mean asynchronous messages generation rate (MGR). For MGR values in the range from 0 to 10^4 msg/s, no deadline misses were experienced by asynchronous messages. From MGR = 1 to 1.67 x 10^4 msg/s the DMR remains below 4%, while for MGR values higher than 1.67 x 10^4 msg/s queues start to be filled up and the network approaches saturation.

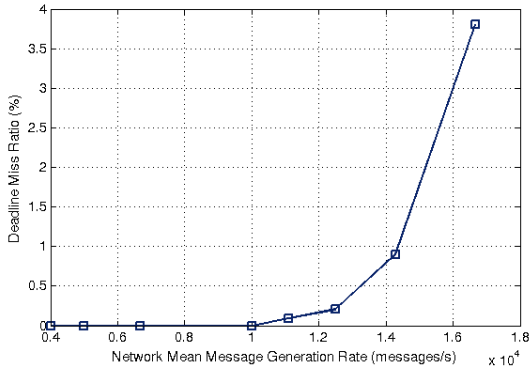

Fig. 5. DMR as a function of the mean message generation rate

These results highlight the ability of the EDF-based Swapping approach to support time constrained traffic with varying messages arrival patterns.

5.1.2. Simulation 2: Comparison with the EtherCAT standard.

The aim of this simulation is to compare the minimum cycle time obtained by the EtherCAT standard and by the EDF-based Swapping approach. In Table 5 the simulation parameters are shown.

For the EtherCAT standard, formulas (1) and (2) allow the minimum cycle time to be obtained, and this value does not vary with the workload.

Symbol	*Value/range*
N	10
S_{PP}	32 bytes
N_{APDU}	from 1 to 8
S_{APDU}	32 bytes (12 header + 20 payload)
t_{sv}	700 ns
DL_{APP}	500 us
APP_{START}	exponential (25 us)
λ	from 150 us to 1000 us

Table 5. Simulation 2 parameters

In the EDF-based Swapping approach the same formulas as EtherCAT are used, but the minimum cycle time varies with the number of Async Type 12 PDUs. Therefore, to perform the comparison, we first have to run simulations with the EDF-based Swapping approach to know the actual number of Async Type 12 PDUs required to cope with the generated workload. This way, from each simulation we obtain the minimum number of Async Type 12 PDUs that provide no deadline misses for a given mean message generation period.

In Fig. 6, on the left side, we compare the actual minimum cycle time of the EDF-based Swapping with the one of the EtherCAT standard.

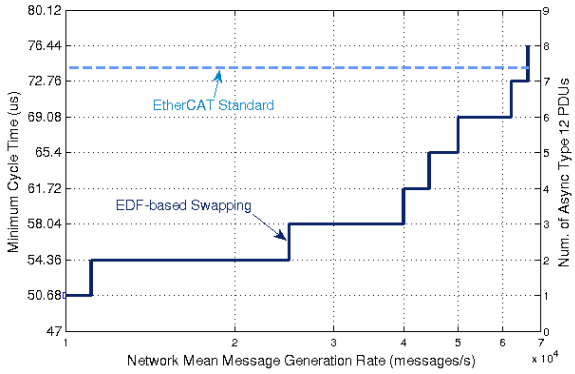

Fig. 6. Comparison of the EDF-based Swapping with the EtherCAT standard for different message generation rates

The minimum cycle time for the EtherCAT standard calculated using formulas (1) and (2) is 74.2 us.

Fig.6 shows that, while the EtherCAT standard has a cycle time with a fixed duration, the EDF-based Swapping approach allows for shorter frames (and thus

shorter cycle times) without any deadline misses. The right side of Fig. 6 shows the number of Async Type 12 PDUs that are needed to have no deadline misses for a given asynchronous real-time workload. When the asynchronous traffic increases, the number of Async Type 12 PDUs required to experience no deadline misses increases, thus also increasing the cycle time. We notice that, for workloads over 6 x 10^4 messages/s, the number of Async Type 12 PDUs that are needed for no deadline misses grows and, in this case, the EtherCAT standard, which handles the asynchronous traffic as periodic one, becomes more convenient than the EDF-based Swapping approach. These results suggest that, when the asynchronous workload is low, EDF-based Swapping is the preferred option, as it provides for shorter cycle times than the EtherCAT standard and therefore offers better support to asynchronous real-time traffic. In particular, applications requiring very short cycle times can be supported.

5.2. Comparison with the CAN-Like Approach

In this simulation, a comparison between the EDF-based Swapping approach and the CAN-Like Approach proposed in [10] is performed. The CAN-Like approach differs from the EDF-based one in various aspects, listed as the following:

- Asynchronous messages have a static priority, which is set offline during the system configuration.
- In order to allow a slave to know if its message won the contention and has arrived at the destination, an additional telegram must be sent from the master. A slave cannot move to the next message to be transmitted until it finds out that its previously transmitted message was the winner[1].

In this simulation, both in the CAN-Like and in the EDF-based Swapping approach, all the asynchronous real-time messages have the same relative deadlines which are chosen with a uniformly distributed probability. The chosen values, i.e., 500, 600, and 700 microseconds are significantly larger than the cycle time for both the protocols under study. For comparison purposes, in the CAN-Like approach different priorities are chosen according to the Deadline Monotonic algorithm, i.e., messages with a smaller relative deadline have a higher priority. This scheduling strategy is very convenient for the CAN-Like approach. In the EDF-based Swapping approach, absolute deadlines are assigned based on both the relative deadline and the message generation times. The DMR versus the mean message generation rate is evaluated.

The simulation parameters for both protocols are specified in Table 6.

[1] This problem was partially resolved in [11] with the introduction of the Multiple Arbitrate (MARBS) Telegrams, where a telegram can contain multiple asynchronous data, but the confirmation message from the master is always required.

Symbol	*CAN-Like*	*EDF-based Swapping*
N	10	10
S_{PP}	32 bytes	32 bytes
N_{APDU}	1	1
S_{APDU}	20 bytes	32 bytes (12 header + 20 payload)
t_{sv}	700 ns	700 ns
DL_{APP}	500, 600, 700 us	500, 600, 700 us
λ	from 1000 to 2400 us	from 600 to 1000 us
CL_{PRIO}	700 (higher), 600, 500	-

Table 6. Simulation parameters for the comparison with the CAN-Like approach.

In Fig. 7 the results of the comparison between the CAN-Like Approach and the EDF-based Swapping Approach are shown. The CAN-Like approach experiences a higher number of deadline misses than the EDF-based Swapping under the same workload. Moreover, for MGR values higher than 1 x 10^4 msg/s with the CAN-Like approach queues start to be filled up and the network approaches saturation, while with the EDF-based Swapping approach saturation starts for MGR values higher than 1.67 x 10^4 msg/s.

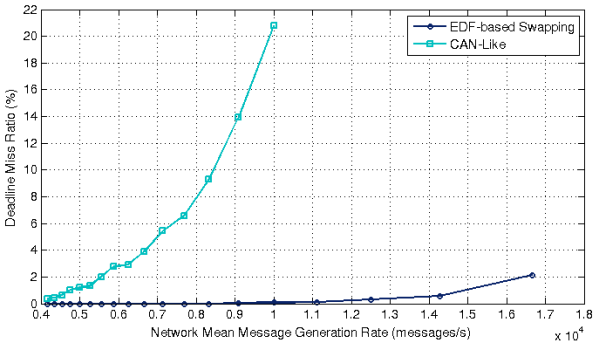

Fig. 7. Deadline miss comparison between the CAN-Like and the EDF-based Swapping Approach

There are two reasons for this result. First of all, the CAN-Like approach requires two cycles for each asynchronous transmission (as the telegram must be acknowledged with another one containing the same data as that received), while the EDF-based Swapping approach requires only one cycle. This explains both the higher number of deadline miss experienced by the CAN-Like approach and also why the EDF-based approach reaches saturation under much higher workloads than the CAN-Like one. Moreover, the high number of deadline miss is also due to starvation, which in the CAN-Like approach affects low priority messages under high workloads.

6. Conclusions and Future Works

The EDF-based Swapping Approach proposed in this paper provides an efficient way to transmit asynchronous real-time messages over EtherCAT networks.

Simulations have proven that the EDF-based Swapping approach significantly reduces the minimum cycle time required for transmitting asynchronous real-time traffic if compared with the EtherCAT standard, in which the asynchronous real-time traffic is scheduled as periodic traffic. This is because Async Type 12 PDUs allow to schedule asynchronous traffic by using EtherCAT frames that are shorter than the ones used in the standard approach. The number of Async Type 12 PDUs embedded in the EtherCAT frame affects the DMR of asynchronous messages, as a lower number of Async Type 12 PDUs entails longer delays for asynchronous real-time traffic. As a result, a suitable tradeoff between the cycle time and the DMR of asynchronous messages depending on the application considered has to be found.

The EDF-based Swapping approach has also been compared with the CAN-Like proposed in [10]. We found that both approaches are suitable for supporting event-driven traffic. However, the EDF-based approach is deadline-aware, so it is more suitable for asynchronous real-time messages. In fact, as the EDF-based Swapping approach explicitly takes deadlines into account, although all the asynchronous messages are given the same chance for transmitting, the message with the closest deadline will progress on the network faster than the ones with less urgent deadlines. On the contrary, in the CAN-Like approach, it is the static priority of the message that rules the contention. Although the priority assignment in the CAN-Like approach can be made so as to reflect the time criticality of the messages, as we did in this paper using a Deadline Monotonic scheduling, however, the typical problems that are found with static priority in real-time scheduling, i.e., the potential starvation for low priority messages under high workloads, may still occur. This is not the case for the EDF-based approach, as even under high workloads, messages with similar deadlines will be dealt with in the same way. As far as alarms are concerned, to be on the safe side the most critical alarms can be scheduled as periodic real-time traffic, while the other ones can be scheduled with the EDF-based Swapping with deadlines assigned by the application layer.

Future works will address a comprehensive assessment of the EDF-based Swapping approach in several different scenarios. Moreover, in order to further improve the performance, mechanisms able to embed more than one APDU in one Async Type 12 PDU, similar to that proposed in [11], will be investigated and a comparison with [11] will be performed. In addition, admission control tests for the EDF-based swapping approach to avoid deadline misses for asynchronous traffic for a given constraint on the cycle time will be investigated. We will also further address alarms handling.

Finally, an algorithm for dynamically changing the number of Async Type 12 PDUs depending on the actual asynchronous real-time workload will be addressed.

7. References

[1] IEC 61158-3-12 Ed. 2, "*Industrial communication networks - Fieldbus specifications - Part 3-12: Data-link layer service definition - Type 12 elements*", 2010.

[2] IEC 61158-4-12 Ed. 2, "*Industrial communication networks - Fieldbus specifications - Part 4-12: Data-link layer protocol specification - Type 12 elements*", 2010.

[3] IEC 61784-2 Ed. 2, "*Industrial communication networks - Profiles - Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*", 2010

[4] S. Vitturi, L. Peretti, L. Seno, M. Zigliotto, C. Zunino, "*Real-time Ethernet networks for motion control*", Computer Standards & Interfaces, 33, 465–476, 2011.

[5] J. Jasperneite, M. Schumacher, K. Weber, "*Limits of Increasing the Performance of Industrial Ethernet Protocols*" in Proc. of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp. 17-24, Patras, Greece, 25-28 Sept. 2007.

[6] G. Prytz, "*A performance analysis of EtherCAT and PROFINET IRT*", in Proc. of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp. 408-415, Hamburg, Germany, Sept. 2008.

[7] M. Knezic, B. Dokic, Z. Ivanovic, "*Topology aspects in EtherCAT networks*", in Proc. of the 14th IEEE International Power Electronics and Motion Control Conference (EPE-PEMC), T1, 1-6, Ohrid, Republic of Macedonia, Sept. 2010.

[8] G.Cena, S. Scanzio, A. Valenzano, C. Zunino "*A Distribute-Merge Switch for EtherCAT Networks*", in Proc. of the 8th IEEE International Workshop on Factory Communication Systems (WFCS), 121-130, Nancy France, May 2010.

[9] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, C. Zunino, "*Evaluation of EtherCAT Distributed Clock Performance*", IEEE Transactions on Industrial Informatics, 8, 1, 20-29, Feb. 2012.

[10] G. Cena, A. Valenzano, C. Zunino, "*An arbitration-based access scheme for EtherCAT networks*", in Proc. of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA), 416-423, Hamburg, Germany, 15-18 Sept. 2008.

[11] G. Cena, I. C. Bertolotti, A. Valenzano, C. Zunino, "*A high-performance CAN-Like arbitration scheme for EtherCAT*", in Proc. of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1-8, Mallorca, Spain, 22-25 Sept. 2009.

[12] A. Di Stefano, A. Gangemi, L. Lo Bello, O. Mirabella, "*Slot swapping mechanisms for Process Control Networks*", in Proc. of the IEEE International Symposium on Industrial Electronics (ISIE), vol. 1, pp 143-148, Guimaraes, Portugal, 07-11 July 1997.

[13] A. Di Stefano, A. Gangemi, L. Lo Bello, O. Mirabella, "*A slot swapping based fieldbus*", in Proc. Of the 24th

Annual Conference of the IEEE Industrial Electronics Society (IECON), vol. 1, pp. 214-219, Aachen, Germany, 31 Aug. - 4 Sept. 1998.

[14] L. Lo Bello , A. Gangemi, "*A slot swapping protocol for time-critical internetworking*", Journal of Systems Architecture, vol. 51, pp. 526-541, Elsevier, 2005.

[15] L. Liu, J. Layland, "*Scheduling algorithms for multiprogramming in a hard real-time environment*", Journal of the ACM (JACM) 20, 1, pp. 46-61, 1973.

[16] OMNeT++ Network Simulator Framework.