# Synthesis of Communication Schedules for TTEthernet-Based Mixed-Criticality Systems

*Abstract*— **Embedded or cyber physical system consist of multiple devices, exchanging data of different criticality levels and still have to meet real-time requirements. This can be accomplished with the so called TTEthernet (TTE) Protocol. TTE differentiates between three criticality levels of traffic: Time-triggered (TT) traffic, Rate-Constrained (RC) traffic (!EVENT TRIGGERED!) and Best-Effort traffic. In the following, we want to create a static schedule for every device in the TTE network, that guarantees that every message meets its deadline and minimizes the end-to-end delay of RC messages.**

## I. Introduction

In the realm of embedded systems, where precise timing and synchronization are paramount, the advent of Time-Triggered Ethernet (TTE) has revolutionized the way data is transmitted and managed. TTE is a deterministic communication protocol that ensures reliable information exchange among interconnected devices in safety-critical applications. One crucial aspect of TTE is the synthesis of static schedules, which plays a pivotal role in guaranteeing the timely and predictable execution of tasks.

Embedded systems are pervasive in our modern world, found in diverse domains such as automotive, aerospace, industrial automation, and medical devices. These systems often operate in real-time environments, where stringent timing requirements must be met to ensure safety, efficiency, and reliability. Real-time embedded systems must process and respond to events within strict deadlines, often measured in microseconds or even nanoseconds. Any failure to meet these timing constraints can lead to catastrophic consequences, endangering human lives and causing significant financial losses.

To address the real-time demands of embedded systems, TTE emerged as a cutting-edge technology. By employing a time-triggered approach, TTE eliminates the uncertainties and non-determinism associated with traditional Ethernet protocols, where data packets contend for network resources, leading to unpredictable delays and potential packet losses. In TTE, the network operates according to a predefined schedule, which dictates when each device can transmit and receive data. This deterministic nature of TTE makes it an ideal choice for safety-critical applications, where the synchronization of activities is paramount.

The synthesis of static schedules in TTE plays a crucial role in achieving reliable and predictable communication within a network. Static schedules are pre-computed timetables that determine the precise timing of events and data transmissions for each device in the network. These schedules are derived from an analysis of the system's requirements, considering factors such as the tasks to be executed, their timing constraints, and the available network bandwidth.

The importance of static schedules in TTE lies in their ability to ensure predictable behavior and timing guarantees for critical tasks. By carefully allocating resources and time slots to different devices, static schedules allow for the prevention of data collisions, latencies, and other timing-related issues. The synthesis process involves sophisticated algorithms and optimization techniques to find an optimal schedule that maximizes resource utilization, minimizes delays, and satisfies the timing constraints of the system.

In summary, the synthesis of static schedules for Time-Triggered Ethernet is a vital area of research and development in embedded systems. It addresses the real-time requirements of safety-critical applications by providing deterministic and predictable communication. By synthesizing static schedules, embedded systems can achieve efficient resource utilization, timely execution of tasks, and reliable data transmission, ultimately contributing to the safety, effectiveness, and success of modern embedded systems.

In section II to section IV we will start with the introduction of the paper "Synthesis of communication schedules for TTEthernet-based mixed-criticality systems" [2].

## II. TTEthernet

In this section, we briefly discuss the basic network design of TTEthernet. We will introduce a small example network to consolidate the concepts and than cover the different types of traffic.

### A. Network Design

A TTE network can be imagined as a graph with interconnected nodes, allowing a multi-hop communication via full duplex links. The nodes are End Systems ($ES$) and Network Switches ($NS$) to which the ESes are connected to. In general a TTE network consists of a set of clusters that can be considered as a smaller network. We are only interested in synthesizing static schedules of these clusters, since synthesizing those for the whole network can become a very hard to solve problem.

A TTE cluster can be formalized as an undirected, since the links are full duplex, Graph $G = (V, E)$ with $V = ES \cup NS$, where $ES$ is the set of end systems and $NS$ is the set of network switches.

The differentiation of traffic of different criticality levels is crucial for synthesizing the static schedules. To accomplish

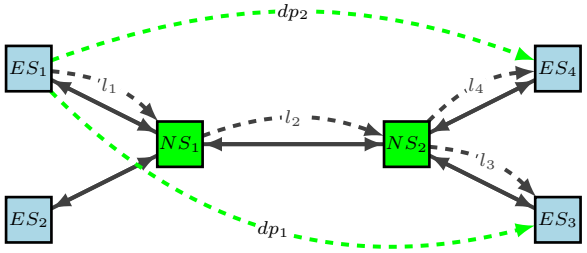


Fig. 1. TTEthernet Cluster

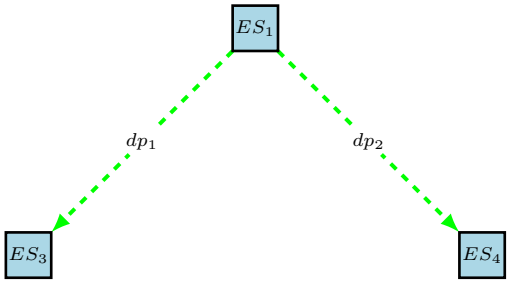


Fig. 2. virtual link $vl_1$

this we have to create a layer of abstraction, from the physical link between devices to a logical sender-receivers connection, the so called virtual links. Each virtual link $vl_i$ is a tree, with the sender as the root and the receivers as the leafs. We denote the set of virtual links with $VL$. To reach the goal of seperating mixed-criticality traffic, each virtual link is assigned a single message. A virtual link $vl_i$ consists of one or multiple dataflow paths, one for each receiver of the message. A dataflow path $dp_i$ is a ordered list of dataflow links and a dataflow link $l_i$ is a directed connection between two directly connected devices. A dataflow link $l_1$ is a ordered tuple of nodes $v_j, v_k \in V$, representing a directed link between two nodes.

Let's consider the following network cluster example of Figure 1. This cluster consists of $ES = \{ES_1, ES_2, ES_3, ES_4\}$ and $NS = \{NS_1, NS_2\}$ as nodes and the physical connection as edges, so the network topology can be depicted as $G = (V, E)$ where $V = ES \cup NS = \{ES_1, ES_2, ES_3, ES_4\} \cup \{NS_1, NS_2\}$

A message $m$ is sent from $ES_1$ and should be transmitted to $ES_3$ and $ES_4$. For this purpose we want to create a virtual link $vl_1$ that represents this exact route from $ES_1$ to $ES_3$ and $ES_4$. Since $vl_1$ has two receivers we need two dataflow paths $dp_1$ and $dp_2$ that represent the end-to-end communication. $dp_1$ is the connection from $ES_1$ to $ES_3$ and consists of the dataflow links $l_1, l_2, l_3$. $dp_2$ is the connection from $ES_1$ to $ES_4$ and consists of the dataflow links $l_1, l_2, l_4$. Putting it together the link topology looks as the following:

$$l_1 = [ES_1, NS_1] \in L \quad (1)$$
$$l_2 = [NS_1, NS_2] \in L \quad (2)$$
$$l_3 = [NS_2, ES_3] \in L \quad (3)$$
$$l_4 = [NS_2, ES_4] \in L \quad (4)$$
$$dp_1 = [l_1, l_2, l_3] = [[ES_1, NS_1], [NS_1, NS_2], [NS_2, ES_3]] \quad (5)$$
$$dp_2 = [l_1, l_2, l_4] = [[ES_1, NS_1], [NS_1, NS_2], [NS_2, ES_4]] \quad (6)$$
$$vl_1 = dp_1 \cup dp_2 \quad (7)$$
$$vl_1 \in VL \quad (8)$$

The message can be sent directly according to the schedule without any further routing process, because of the predetermined static schedules.

### B. Frames

As mentioned before, TTE supports three types of traffic, time-triggered ($TT$), rate-constrained ($RC$) and best effort ($BE$). For synthesizing the schedules we only use $TT$ and $RC$ messages, since $BE$ messages have no end-to-end delay constraints, they are not that important and therefore have the least level of criticality. Each message $m_i$ is packed into one frame $f_i$ and we call the set of all frames $F = F^{TT} \cup F^{RC} \cup F^{BE}$. Each frame $f_i$ can contain exactly one message $m_i$. Each frame $f_i \in F^{TT} \cup F^{RC}$ has the following attributes:

- $f_i.size$: The size of a frame
- $f_i.period$: The time period, when the next instance of this frame is sent
- $f_i.deadline$: The minimum end-to-end delay

For RC frames, we also define the rate $f_i.rate = 1/f_i.period$ of a frame $f_i$.

The mentioned instance of a frame, is denoted with $f_{i,x}$ for the xth-instance of the frame $f_i$. With its instance we can distinguish, to which period a frame $f_i$ belongs.

We denote this mapping function of frames onto virtual links with:

$$M : F \to VL; M(f_i) \mapsto vl_i \quad (9)$$

Since the speed of all physical connections and the size of the frames are given, the transmission duration on a dataflow link $l_i = [v_j, v_k]$ for a frame $f_g$ is denoted as $C_g^{l_i} = C_g^{[v_j, v_k]}$.

### C. Time-Triggered Transmission

In the following we are going to discuss the transmission of TT frames in a TTEthernet network. We introduce some basic concepts and then look at it more closely with an example.
Each node $v_j \in V$ (ESes and NSes) has its own receiving and sending schedule $s_{j,r}, s_{j,s}$, from which we can derive which frames $f_i \in F^{TT}$ will pass the node. For all ESes, we create a sending Buffer $B_{i,Tx}$ if a TT frame $f_i$ is in the sending schedule $s_{i,r}$. Vice versa for the receiving schedule. Since a End System is either a sender or a receiver of a frame, a End System will contain utmost one Buffer for a frame $f_i$.

For all NSes we create a Buffer $B_{i,Tx}$ if a frame $f_i$ is in the schedule $s$. We only need to look through the sending or receiving schedule, since frames are just relayed by a NS and therefore are in both schedules.
Each node $v_j$ has a scheduler task $TT_s$ that sends the frames, stored in the sending buffer $B_{i,Tx}$, according to the sending schedule $s_{j,s}$. For the receiving process, each node has a filtering unit $FU$. The $FU$ of a network switch checks the integrity and validity of the received frame $f_i$ and forwards it to the TT receiver task $TT_R$, which checks if the frame is received in the specified time window of the receiving schedule $s_{j,r}$. If so, the TT receiver task stores the frame $f_i$ in the buffer $B_{i,Tx}$, otherwise the frame is dropped. The $FU$ of an end system also checks integrity and validity of the received frame and then stores it directly in the buffer $B_{i,Rx}$. There is no further comparison with the receiving schedule needed, since the received frame is on its destination and will not lead to unnecessary traffic in the network.
Lets illustrate the concept of a TT message with the example of figure 3: The steps of the TT frame are labeled with the letters a-j. In $ES_1$ a task $\tau_2$ sends a message $m$ to $ES_2$. Therefore the message is packed into a frame $f_1$ and stored in the corresponding buffer $B_{1,Tx}$ (b). The scheduler task $TT_s$ sends the frame in the Buffer $B_{1,Tx}$ (d), according to the send schedule $s_{1,s}$ (c). After the frame is sent on a dataflow link to $NS_1$ (e), the filtering unit $FU$ checks the received frame (f). If the frame is fine, the receiver task $TT_R$ checks (h), if it is received before its deadline $f_1.deadline$, according to the receiving schedule $s_{1,r}$ (g). If the frame is accepted, the receiver task stores the frame in the buffer $B_{1,Tx}$. The scheduler task $TT_S$ of $NS_1$ will send the frame according to the send schedule (j) to its destination $ES_2$. There comes the check of the filtering unit again (k), after which it is stored in the Buffer $B_{1,Rx}$, from where it can be used by a another task of the ends system.

### D. Rate-Constrained Transmission

Rate-Constrained traffic contains event-triggered messages, therefore it is not as deterministic as time-triggered transmissions. We have to assume that there can be sent frame instances of the same frame in bursts (fast, one after another). Hence, if we want to schedule them, we define a "cooldown", when the next frame instance of this frame is allowed to get sent. We call this "cooldown" the Bandwith Allocation Gap (BAG). The BAG for a virtual link $vl_i$ and a RC frame $f_j$ is defined as $BAG_i \leq 1/f_j.rate$
Since RC traffic has a lower priority than TT traffic, we need to ensure that RC messages are only sent, when there is no TT traffic scheduled and need a strategy what to do with RC messages that might interfere with scheduled TT messages. We have three possible solutions for this problem:

a) shuffling: The TT frame is delayed, until the transmission of the RC frame is finished.
b) timely block: If the transmission of a TT frame would start, before the transmission of a RC frame has ended, the RC frame will be postponed. Since we use static schedules, we can easily check, if there are any TT frames to which this scenario applies, when we want to send a RC frame.
c) pre-emption: A currently transmitting RC frame would be pre-empted (stopped, due to another frame), when there is a TT frame scheduled. The transmission of the RC frame would be started immediately after the transmission of the TT frame has finished.

In the following examples we will only use the timely-block b) method.

We will now discuss the way, how RC messages are sent from one end system to another and then have a closer look at an example.
For each virtual link $vl_i$ we create a queue. We create queues with the same procedure as we did for buffers, so we create a queue $Q_{i,Tx}$ for sending a frame $f_i$ and a queue $Q_{j,Rx}$ for receiving a frame $f_j$. For each sending queue $Q_{i,Tx}$ in an end system we have a traffic regulator task $TR_i$ that ensures that the BAG of the frames is respected.
The RC scheduler task $RC$ does the multiplexing of RC frames coming from the regulator tasks and sends the multiplexed frames to the scheduler task $TT_S$ that integrates the RC traffic into the TT traffic, according to the chosen integration policy. The filtering unit $FU$ does also check the integrity and validity of RC frames. Network switches have a traffic policing task (TP) that checks, if the BAG for two frame instances $f_{i,j}, f_{i,k}$ of the the same frame $f_i$ is respected. If the measured interval is shorter than the defined BAG, the latter frame instances is dropped. The advantage of the TP is that it can compensate a missbehaving ES that sends RC messages in bursts. Since we introduced all RC specific parts of the network, we will apply the theory on an example frame, sent from $ES_1$ to $ES_2$ over $NS_1$. The steps in the figure 3 of the RC frame are depicted with the numbers 1-12.
The end system $ES_1$ wants to send a RC message m, therefore it packs it into the frame $f1$ (1) and inserts it into the outgoing queue $Q_{1,Tx}$ (2). The traffic regulator task $TR_1$ reads from this queue, and forwards them to the traffic regulator task $RC$ with respect to the defined $BAG_1$ (3). The $RC$ multiplexes the frame $f_1$ with all other frames that were forwarded by other traffic regulator tasks (4). Then the scheduler task $TT_S$ sends the frame according to the chosen integration policy (5).

## III. The Problem

### A. Description

TTEthernet is applied for static networks, so the network topology $G$ is predetermined. We also know the exact sending times of TT frames $F^{TT}$ and the maximum end-to-end delay of TT and RC frames $F^{TT} \cup F^{RC}$. Having all this information of the network leads to more determinism, since we know the topology and the source and destination of the frames, we can apply static schedules, instead of a classic routing mechanism. This gives us the advantage of increasing the speed and predictability, what makes it extremely useful for real-time applications.

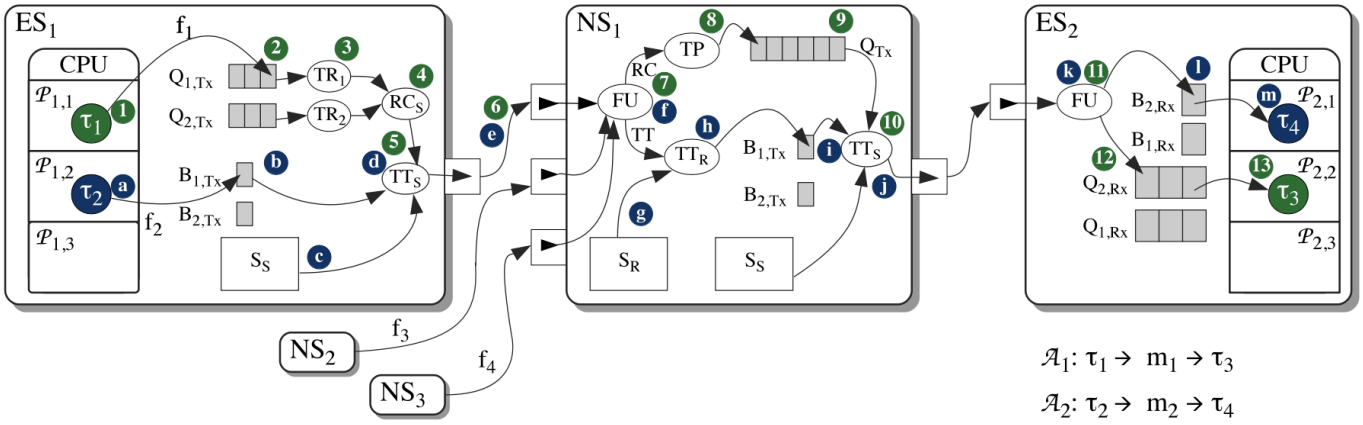


Fig. 3. TT and RC traffic example [2]

The problem is to create the schedules $s_i$ for all nodes $v_i \in V$ in the network, but since the problem is NP-complete [1] we most probably won't find the best solution
The TTESO strategy is subdivided in two main parts:

- initial solution: We want to create a first valid solution, which we then use in the second step for further optimization. For this purpose we use a greedy algorithm called "List Scheduling" (LS).
- final solution: Starting with the initial solution, we now transform the schedules in different ways to reduce the end-to-end delay, with a algorithm called "Tabu Search" (TS).

*1) List Scheduling:* For this algorithm, we want the frames represented with their temporal constraints. Therefore we use the dataflow links $l_i$ of a virtual link $vl_j$ as nodes and use the directed edges to represent the temporal constraints. We depict with $f_{i,x}^{l_j}$ the xth-instance of the frame $f_i$ on the dataflow link $l_j$. This representation allows us to not only represent the dataflow links of a virtual as a tree (for this purpose, the tree representation of a virtual link itself would have been sufficient), but to show exactly when which frame instance is on which dataflow link. With the example network of figure 1, this gives us the tree, shown in figure 6 for the frame instance $f_{1,1}$ transmitting from $ES_1$ to $ES_3$ and $ES_4$ over the virtual link $vl_1$ (figure 2).
To apply List Search, we now merge all trees of all frame instances together in one graph $K$. We connect the root of each tree to a dummy source node in $K$ and the leafs of each tree to a dummy sink node in $K$. The result, using the network topology of figure 5 and two periods of the frames $f_2, f_3 \in F^{TT}, f_1 \in F^{RC}$ looks like the figure 4.
List Search takes as input parameters the network topology $G$, the TT and RC frames $F^{TT} \cup F^{RC}$, the virtual links $VL$, the mapping of frames onto virtual links $M$ and the just declared graph $K$. LS tries to create a schedule, based upon the graph $K$, representing the time constraints, that meets the frame deadlines. We denote this initial set of schedules $S^o$. The result of the initial schedules, using the topology

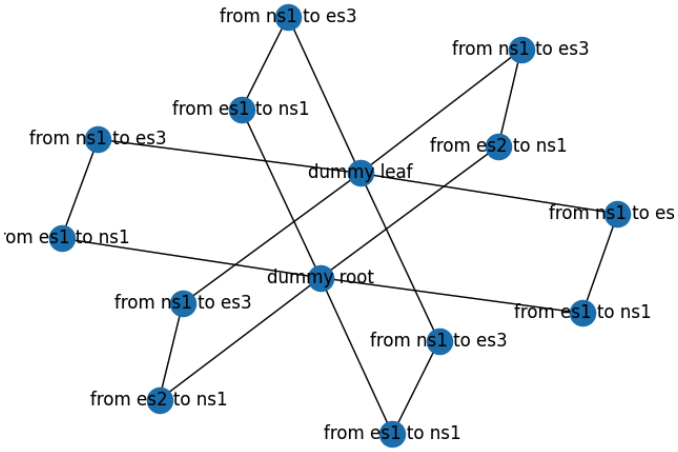


Fig. 4. Merged virtual link trees of figure 5

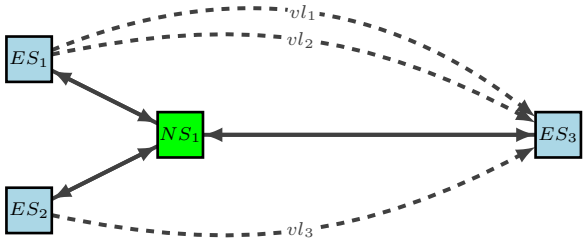


Fig. 5. TTEthernet small cluster

of figure 5 with the frames $f_2, f_3 \in F^{TT}, f_1 \in F^{RC}$ and mapping $M(f_2) = vl_1, M(f_3) = vl_3, M(f_2) = vl_1$, is shown in table I.

*2) Tabu Search:* Tabu Search (TS) starts with the initial solution $S^o$ and applies different kind of changes to improve the previous solution. The quality of the solution is measured by a cost function:

$$cost = w_{TT} \times \delta_{TT} + w_{RC} \times \delta_{RC} \tag{10}$$

TABLE I
LS - INITIAL SOLUTION

| **Dataflow Link** | **schedule** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $[ES_1, NS_1]$ | $f_{2,1}$ | $f_{1,1}$ | | | $f_{2,2}$ | | $f_{1,2}$ | | | ... |
| $[NS_1, ES_3]$ | | $f_{2,1}$ | $f_{3,1}$ | $f_{1,1}$ | | $f_{2,2}$ | | $f_{3,2}$ | $f_{1,2}$ | ... |
| $[ES_2, NS_1]$ | $f_{3,1}$ | | | | | | $f_{3,2}$ | | | .. |

using the network topology of figure 5

TABLE II
TS - POSTPONE MOVE ON $f_{2,1}^{[ES_1,NS_1]}$

| **Dataflow Link** | **schedule** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $[ES_1, NS_1]$ | | $f_{1,1}$ | $f_{2,1}$ | | $f_{2,2}$ | | $f_{1,2}$ | | | ... |
| $[NS_1, ES_3]$ | | $f_{2,1}$ | $f_{3,1}$ | $f_{1,1}$ | | $f_{2,2}$ | | $f_{3,2}$ | $f_{1,2}$ | ... |
| $[ES_2, NS_1]$ | $f_{3,1}$ | | | | | | $f_{3,2}$ | | | .. |

using the network topology of figure 5

TABLE III
TS - SATISFY CONSTRAINTS

| **Dataflow Link** | **schedule** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $[ES_1, NS_1]$ | | $f_{1,1}$ | $f_{2,1}$ | | $f_{2,2}$ | | $f_{1,2}$ | | | ... |
| $[NS_1, ES_3]$ | | | $f_{3,1}$ | $f_{1,1}$ | $f_{2,1}$ | $f_{2,2}$ | | $f_{3,2}$ | $f_{1,2}$ | ... |
| $[ES_2, NS_1]$ | $f_{3,1}$ | | | | | | $f_{3,2}$ | | | .. |

using the network topology of figure 5

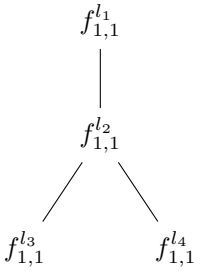


Fig. 6. Tree visualization of temporal constraints

Where $\delta$ is the degree of schedulability, calculated as follows:

$$\delta_{TT/RC} = \begin{cases} c_1 = \sum_i max(0, R_{f_i} - f_i.deadline) & \text{if } c_1 > 0 \\ c_2 = \sum_i (R_{f_i} - f_i.deadline) & \text{if } c_1 = 0 \end{cases} \quad (11)$$

We denote the worst case end-to-end delay of a frame $f_i$ with $R_{f_i}$. If there is at least one frame $f_i$ that does not meet its deadline $f_i.deadline$, $c_1$ will be larger than 0, so we use $c_1$. Otherwise, if all frames are schedulable, we use $c_2$.
If a frame $f_i$ is not schedulable, its corresponding weight is set to a very high number, to signal the algorithm that this solution is definitely worse than the last. Otherwise the weight can be chosen by the designer.
The changes, TS can apply are called "moves". A move is applied to the current solution, then the cost function is evaluated and compared to the previous value. If it is better than the previous, the new solution is stored as the best known solution. TS stores previous solutions in a "Tabu list", to prevent loops and local minima.

There are four types of moves that can be applied to TT frame instances:

i) advance: Move the scheduled send time of a frame instance $f_{i,x}$ on a node $v_j$ to an earlier moment.
ii) advance predecessors: Move the scheduled send time of all frame instances in the set of predecessors of a frame instance $pred(f_{i,x}^{l_k})$ to an earlier moment.
iii) postpone: Move the scheduled send time of a frame instance $f_{i,x}$ on a node $v_j$ to a later moment.
iv) postpone successors: Move the scheduled send time of all frame instances in the set of successors of a frame instance $succ(f_{i,x}^{l_k})$ to a later moment.

For each move we apply we have to make sure that the time constraint is not violated. If executed an advance move, we have to make sure that the new scheduled send time is not before the frame sent by a task. For postpone moves, we need to ensure that the new receiving time is before the deadline of the moved frame.
In table II, we moved the TT frame $f_{2,1}^{[ES_1,NS_1]}$ to a later moment in time, we used an advance move. Now we have to check that there is no successor of the moved frame $succ(f_{2,1}^{[ES_1,NS_1]})$ that is scheduled before $f_{2,1}^{[ES_1,NS_1]}$. Since this is the case for $f_{2,1}^{[NS_1,ES_3]}$ (line 3, column 3), we need to satisfy the constraints again. The solution of this is shown in table III

### *B. Worst-case end-to-end delay*

The worst-case end-to-end delay for a RC frame $fi \in F^{RC}$ is given by the sum of maximum delay of the queue $Q_{f_i}^{[v_j,v_k]}$ in each node $v_j \in V$ plus the transmission duration $C_{f_i}^{[v_j,v_k]}$

on each dataflow link $l_k = [v_j, v_k] \in vl_i$, the frame passes. The superscript on the queue $Q$ that denotes a dataflow link $l_x = [v_j, v_k]$, stands for the queue at node $v_j$.

$$R_{f_i} = \sum_{l_x=[v_j,v_k] \in vl_i} (Q_{f_i}^{[v_j,v_k]} + C_{f_i}^{[v_j,v_k]}) \tag{12}$$

The worst-case queuing delay $Q_{f_i}^{[v_j,v_k]}$ can be calculated, by adding up the queueing delay that results in scheduled TT frames $Q_{f_i,[v_j,v_k]}^{TT}$ that may (depending on the chosen method in **??** postpone the RC frame $f_i$, the delay caused by other RC frames that are placed before $f_i$ in the queue $Q_{f_i,[v_j,v_k]}^{RC}$ and the technical latency $Q_{[v_j,v_k]}^{TL}$ that is caused by the hard- and sofware of the node $v_j$.

$$Q_{f_i}^{[v_j,v_k]} = Q_{f_i,[v_j,v_k]}^{TT} + Q_{f_i,[v_j,v_k]}^{RC} + Q_{[v_j,v_k]}^{TL} \tag{13}$$

## IV. Problems during the implementation

The biggest problem I encountered during the implementation of the TTESO algorithm was clearly choosing the moves. The report states the following: "...these "moves" are not performed random, but chosen based on a candidate list of moves that may improve the search." This was obviously no help at all, since the paper does not include the information on how to create this candidate list of moves. In my implementation I decided to choose the moves random, but for small clusters, like the examples, trying all moves and only going on with the best (the solution with the least cost) is also feasible.

## V. Working with AI

After reading the paper, I tried using ChatGPT to get some additional information about TTEthernet and the scheduling methods. Unfortunately, the answers were too general to bring them in the report and the worst, ChatGPT gave no sources to its statements. I think the developers did this to protect the website owner, in case ChatGPT generates untrue, offending etc statements and put their site below it, although the information were from else where or taken out of context. So for the part related/ further work, it is in my opinion quite useless.

On the other hand, creating figures in latex works quite fine. The first versions of figure 1, figure 2 and figure 6 were generated by ChatGPT. figure 1 and figure 2 needed some changes, but the foundation was quite good. It really helped in being faster, since I did not work a lot with latex before, especially with the module tikz.

I also could not use AI to write scientific texts, since, as mentioned above, the output is too general. To get the output you want, you would have to enter a lot of stuff, so the AI has enough knowledge of the problem. In my case that would have been the whole paper, since ChatGPT seemed to know another paper that cares about static schedule synthesis of the same author [4], but did never use the method described in this one.

Because of this issue, outsourcing the implementation to AI also failed. I used it to automatic create the classes for the network devices and all types of links, but they included just a few attributes and methods to send frames. The implementation of the different link types did all look quite similar, so the only use was to use them as a foundation. The same result could have been achieved with auto-completion tools of an IDE. Nevertheless I like to use AI for coding, because it is very powerful for small scripts that only need knowledge that is "well spread" in the internet, since a lot of people use them/ the same functionality.

Another very good use-case was the introduction, since what you want there are general information about the topic. Because my English skills are limited, it helped me a lot in formulating eloquent sentences, with a good vocabulary.

Using AI for a scientific purpose can be quite hard, since generating high quality and reliable output for a problem with poor data or a even not known problem can be a real challenge.

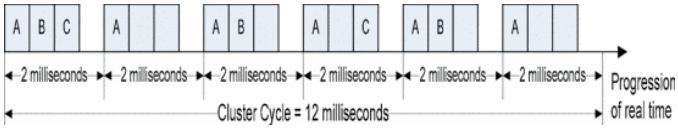


Fig. 7. Cluster Cycle [6]

## VI. Related work

### A. SMT based schedule synthesis

Another way of synthesizing static schedules for TTE is via Satisfiability Modulo Theories (SMT) [4]. This method has the same disadvantage of generating the schedules off-line, but uses a different approach. Problems can be formulated as a SMT, to check if they are satisfiable. In our case we need to convert our problem, for example the timing constraints of each frame, in first-order logic to generate static schedules. This problem formulation can now be solved by a SMT solver, e.g. Yices [3], to check if a valid solution can be obtained. If so, the Yices solver can show a valid variable assignment, what can be interpreted as our static schedules.

### B. Cycle based schedule synthesis

A third method of synthesizing schedules is proposed by Suethanuwong [6]. In this paper, the time is divided in cluster cycles for each cycle. The length of the cycle is calculated by the least common multiple (LCM) of all TT frames $f_i \in F^{TT}$. For example: We have three TT frames $A, B, C$ and their corresponding periods $2, 4, 6$ then the LCM is $LCM(2, 4, 6) = 12$. So a possible pattern would look like figure 7, which is repeated over and over again. To create more space for RC and BE messages, offsets are established that lead to a "remaining time" window during a period. This time window can be used for further TT traffic, if existing, but also for RC and BE traffic.

## VII. TTE vs TSN

Time Sensitve Networking (TSN) is another type of network that allows real time communication and is standardized by the IEEE. It has the advantage of a more precise clock synchronization method (IEEE 1588 Precision Time Protocol)

and sends redundant messages over mulitple paths, to be more resistant against packet loss [7]. If the network topology changes, it can adopt dynamically, without the need of synthesizing static schedules again.
TSN realises this, by creating a time windows, in which only frames of a certain priority can be sent [8]. The priority of a frame is set via VLAN (802.1Q). This has the clear advantage of no need of precomputing and using already existing, well established technologies, like VLAN, since TSN is developed by the IEEE.
Steiner [9] proposed a way to create off-line static schedules for TSN, with a adopted SMT based procedure, but there were no comparison in end-to-end delay to the method of dynamically created schedules.

## VIII. CONCLUSION

In this paper we discussed a method of synthesizing static schedules for TTE, proposed by Steiner [2]. Therefore we mapped RC and TT traffic onto logical end-to-end connection, to partition frames of different criticality levels. To create an initial solution, we use List Search that uses the network topology, the frames, and the mapping of frames onto virtual links, to get a schedule for each network device that satisfies the time constraints. In the second step, we use Tabu Search that uses moves to transform the initial schedules, to reduce end-to-end delays.
Even though the results look promising, TSN seems to have prevailed over TTE, since it uses well established technologies, is more flexible and maintained by the IEE. Another indication for this trend is the quite good research foundation of TSN compared to TTE.

## REFERENCES


[1] J. D. Ullman. NP-complete scheduling problems. J. Comput. Syst. Sci., 10(3):384–393, 1975.
[2] Domitian Tamas-Selicean, Paul Pop, Wilfried Steiner, "Synthesis of communication schedules for TTEthernet-based mixed-criticality systems", https://dl.acm.org/doi/10.1145/2380445.2380518; 2012
[3] Bruno Dutertre, Leonardo de Moura, "The YICES SMT Solver", http://gauss.ececs.uc.edu/Courses/c626/lectures/SMT/tool-paper.pdf; 2006
[4] Wilfried Steiner, "An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks", https://ieeexplore.ieee.org/abstract/document/5702246; 2010
[5] Francisco Pozo, Guillermo Rodriguez-Navas, Hans Hansson, Wilfried Steiner, "SMT-based synthesis of TTEthernet schedules: A performance study", https://ieeexplore.ieee.org/abstract/document/7185055; 2016
[6] Ekarin Suethanuwong, "Scheduling time-triggered traffic in TTEthernet systems", https://ieeexplore.ieee.org/abstract/document/6489749; 2012
[7] Norman Finn, "Introduction to Time-Sensitive Networking", https://ieeexplore.ieee.org/abstract/document/8412458; 2018
[8] John L. Messenger, "Time-Sensitive Networking: An Introduction", https://ieeexplore.ieee.org/abstract/document/8412459; 2018
[9] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelik, Wilfried Steiner, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks", https://dl.acm.org/doi/abs/10.1145/2997465.2997470; 2016