Routing and Queue Assignment in TSN using Peristaltic Shapers

Reza Barzegaran and Paul Pop
DTU Compute, Technical University of Denmark
Email: {mohba,paupo}@dtu.dk

September 14th, 2021

Abstract

This document presents the optimization project for the "02229 Systems Optimization" course at DTU Compute. All the required materials mentioned in the text, including the test cases, are available via DTU Learn. IEEE 802.1 Time-Sensitive Networking (TSN) is a communication standard used in many application areas. Given a set of communication flows, and a TSN network topology consisting of switches and communication links, the problem is to determine the routing of the communication flows and the assignment of flows to the queues of switches, such that the latencies are minimized. This is an intractable combinatorial optimization problem that has to be solved using the techniques presented in the course.

1 Introduction

Motivation: *Mixed-criticality* cyber-physical systems have functions with different safety-criticality requirements, e.g., highly critical, mission critical, and non-critical requirements. For example, a network backbone in a modern vehicle has to integrate Advanced Driver Assistance Systems (ADAS) functions, which rely on high-bandwidth data from sensors, such as video cameras and Light Detection And Ranging (LIDAR), with powertrain functions that have tight timing constraints but use small frame sizes, and diagnostic services, which are not time-critical. Owing to the increase in complexity, and the need to reduce costs, such mixed-criticality applications are currently implemented in *integrated architectures*, where the functions of different criticality share the same distributed platform.

There are various communication protocols on the market, depending on the application area, e.g., FlexRay for automotive, ARINC 664 p7 for avionics [1], and EtherCAT for industrial automation [7]. However, emerging applications, such as ADAS, autonomous driving, or Industry 4.0, have increasing bandwidth demands. For instance, autonomous driving requires data rates of at least 100 Mbps for graphical computing based on camera, radar, and LIDAR data, whereas CAN and FlexRay only provide data rates of up to 1 Mbps and 10 Mbps, respectively. Furthermore, there have been many safety-critical protocols proposed, though only a few of them can support the separation required by mixed-criticality messages [9].

TSN: The well-known networking standard IEEE 802.3 Ethernet [5] meets the emerging bandwidth requirements for safety-critical networks, besides remaining scal-

able and cost-effective. However, Ethernet is unsuitable for real-time and safety-critical applications [3]. Many extensions, such as EtherCAT [7], PROFINET [4], ARINC 664p7 [1], and TTEthernet [10], have been suggested and used in the industry. Although they satisfy the timing requirements, they are mutually incompatible, and hence, they cannot operate on the same physical links in a network without losing real-time guarantees. Consequently, the IEEE 802.1 Time-Sensitive Networking task group [6] has been working since 2012 to standardize the real-time and safety-critical enhancements for Ethernet. TSN primarily consists of amendments ("sub-standards") to IEEE 802.1Q[1]. TSN is quickly becoming the de facto standard in several areas, e.g., industrial, automotive, avionics, space, with a wide industry adoption and several vendors developing TSN switches.

Optimization problem: In this project we are interested in cyber-physical systems which use IEEE 802.1 Time-Sensitive Networking (TSN) for communication, see [8] for a high-level presentation of TSN. TSN-based systems are composed of end systems (ESes) interconnected by network switches (SWs) and duplex physical links, see Section 2.1 for the model of a TSN network.

TSN supports the convergence of multiple traffic types, i.e., critical, real-time, and regular "best-effort" traffic within a single network, and hence, is suitable for mixed-criticality applications. We consider that the applications send messages using the Cycle Specified Queuing and Forwarding (CSQF) [2], which is an extension of the Cycle Queuing and Forwarding (CQF) standard IEEE 802.1Qch-2017 that introduces a "peristaltic shaper', see [8] for an introduction. There are many traffic types that can be used with TSN, in isolation or in combination. The advantage of CSQF is that it requires less effort for the network configuration, while at the same time providing real-time guarantees, i.e., the worst-case latencies of flows can be easily determined.

SWs are forwarding messages from their input (ingress) ports to their output (egress). Each egress port has eight priority queues where messages are placed before transmission. CSQF uses three of the high-priority queues to forward the messages, see Section 2.2 for an explanation on how CSQF works. The mixed-criticality real-time applications are modeled as a set of periodic flows. For each flow, we know its source and destination ESes, its period and its deadline. See Section 2.3 for the details of the application model.

The optimization problem can be formulated as follows, see its formal definition in Section 3. As an input to the problem we have a TSN topology and a set of applications modeled as periodic flows. For each flow, we are interested to determine (i) its route from source to the destination and (ii) the assignment of the flow to the CSQF queues in each SW along its route. We are interested in solutions such that all the flows are schedulable (their deadlines are satisfied) and the mean bandwidth utilization, as defined in Section 4.3, is minimized.

2 System Models

The system model consists of an architecture model, a CSQF switch model, and an application model described in Section 2.1, Section 2.2 and Section 2.3, respectively. Table 1 summarizes the notation.

[1] The references for all sub-standards can be easily found based on their names. All the relevant materials for the project are also available via DTU Learn

2.1 Architecture Model

The network contains several end systems (ESs) that are connected to each other via network switchs (SWs). An ES is either the source (talker) or the destination (listener) of an application flow, whereas a switch forwards the frames of flows. The ESs perform computation of operation.

We determine a time period of C, let's call it a hypercycle, in which the network behavior is cyclic, i.e. the network behavior is the same. We split the hypercycle into several cycles c with the same length $|c|$. The number of cycles in a hypercycle is denoted by $|C|$. Without the loss of generality we assume that the starting of cycles at the different nodes is the same and there is no offset. We assume a cycle length $|c|$ of 12 μs. The hypercycle C should be set to the least common multiple (LCM) over the periods of all flows. So, if you have 3 flows, with the period of 25 milliseconds (ms), 40 ms and 100 ms, then the hypercycle is the LCM of 25, 40, 100, which is 200 ms.

Table 1: Summary of the notation

Notation	Definition	XML tag
C	Hypercycle	–
c	Cycle	–
$\|c\|$	Cycle length	–
$\mathcal{G}$	Network graph	Architecture
$\mathcal{V}$	Graph nodes (ES & SW)	–
v_i	Graph node	Vertex
$v_i.P$	Set of egress ports	–
p_j	A port	–
N	Number of priority queues	–
N_{DN}	Number of CQSF queues	–
$\mathcal{E}$	Set of links	–
$\varepsilon_{i,j}$	A link	Edge
$\varepsilon_{i,j}.s$	Link bandwidth	BW
$\varepsilon_{i,j}.d$	Link delay	PropDelay
$\varepsilon_{i,j}.S$	Link capacity	–
$\varepsilon_{i,j}.D$	Link induced delay	–
$\mathcal{S}$	Set of flows	Application
s_i	Flow	Message
$s_i.v_s$	Talker node	Source
$s_i.v_d$	Listener node	Destination
$s_i.c$	Size	Size
$s_i.t$	Period	Period
$s_i.d$	Deadline	Deadline
$s_i.\mathcal{A}$	Arrival pattern	–
$s_i.\mathcal{D}$	Acceptable deadline	–
$\mathcal{R}$	Set of routes	–
r_i	Route	–
r_i^j	j^{th} link assignment	–
$r_i^j.\varepsilon$	Associated link	–
$r_i^j.q$	Associated queue number	–

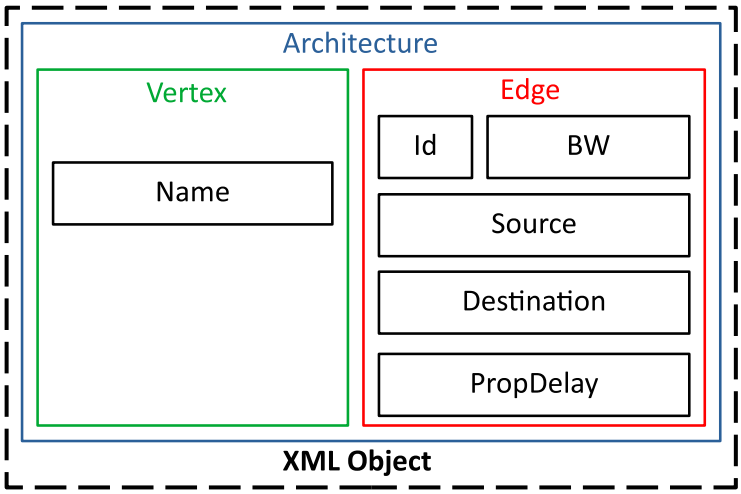

Figure 1: Architecture Input Structure

You may also need to round up the hypercycle to consider the cycle length $|c|$ of 12 μs. For example, 200 ms is 16666.66 cycles of 12 μs, which need to be rounded up to 16667 cycles.

We model the architecture as a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \mathbf{ES} \bigcup \mathbf{SW}$ is the set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. A vertex $v_i \in \mathcal{V}$ represents a node in the architecture which is either an end-system or a network switch. Nodes have input (ingress) and output (egress) ports. We denote the set of egress ports of a node with $v_i.P$. A port $p_j \in v_i.P$ is linked to at most one other node. The set of edges $\mathcal{E}$ represents bi-directional full-duplex physical links. Thus, a full-duplex link between the nodes v_i and v_j is denoted with both $\varepsilon_{i,j} \in \mathcal{E}$ and $\varepsilon_{j,i} \in \mathcal{E}$; a link is attached to one port of the node v_i and one port of the node v_j.

Each link $\varepsilon_{i,j}$ is characterized by the tuple $\langle s, d \rangle$ denoting the bandwidth of the link in Mbit/s and the propagation delay of the link. The propagation delay of a frame on a link $\varepsilon_{i,j}.d$ is calculated based the physical medium, and the link length.

Input Format: The architecture is given as an XML file with the tags denoting the architecture model. The structure of the XML file and the corresponding tags are shown in Fig. 1. The relevant notation to the tags are presented in Table 1.

2.2 CSQF Switch Model

In the introduction we have motivated the use of Cycle Specific Queuing and Forwarding (CSQF) mechanism [2] as a promising standard from the IETF DetNet group. Here we model the details of a CSQF switch needed to formulate our problem. For further details on how CSQF works, the reader is directed to the respective standards.

A CSQF switch (see Fig. 2) consists of ingress ports, a routing function, a routing table, and egress ports. Each egress port of a CSQF switch has a set of N priority queues (typically 8) out of which N_{DN} (by default 3) queues are reserved for time-sensitive traffic and are controlled by a port control function. The N_{DN} time-sensitive queues serve in a round-robin fashion. Thus one active queue is open for transmission and closed for reception, and the $N_{DN} - 1$ inactive queues is only open for reception. A queues is active for a cycle.

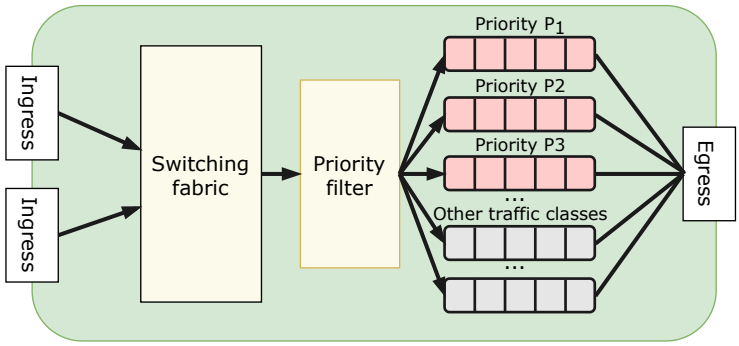

Figure 2: TSN switch internals

The switching fabric receives flows from the ingress ports and forwards each flow to the corresponding egress port considering its Segment Routing ID (SID), i.e. the label contained in the header of the packet. In order to do so, each CSQF switch has a pre-defined static routing table which specifies a range of SIDs for each egress port.

Once a flow is received in an egress port, the port control function uses the flow SID to determine which queue transmits the flow. A cycle table and a queue table are associated statically to each queue for controlling the flow transmission. The cycle table specifies how many cycles each flow should be delayed for transmission relative to the time it has been received in the queue. The queue table specifies the associated queue for each SID.

According to the IEEE 802.1Qch standard, the deterministic transmission of traffic is regulated by the SID tables, i.e., routing tables, cycle tables, and queue tables which contains the when and where a flow with a specific SID must be forwarded. However the IEEE 802.1Qch standard has simplified the cycle table by assigning fixed cycles to the queues, i.e. the queue number 1 is assigned with one cycle, the queue number 2 is assigned with two cycles and the queue number 3 is assigned with three cycles (for ports with default 3 queues reserved for time sensitive traffic).

As discussed earlier each port $p_j \in v_i.P$ is attached to a links originating from the node v_i. Thus the link can also be used alternatively to point out the specific port.

2.3 Application Model

Our model consists of a set of flows denoted with $\mathcal{S}$, which are *real-time* and needed to be routed through the network.

Each flow $s_i \in \mathcal{S}$ is responsible for sending the frames that encapsulate the data and it is characterized by the tuple $\langle v_s, v_d, c, t, d \rangle$ denoting the source node $v_s \in \mathcal{V}$, the destination node $v_d \in \mathcal{V}$, the size in bytes, the period in μs and the flow deadline, i.e., the maximum allowed end-to-end delay in μs.

The flow s_i is transmitted via the route $r_i \in R$, where $\mathcal{R}$ is a set of routes. The route r_i is an ordered list of link assignments, where we denote the j^{th} link assignment with r_i^j. Each link assignment r_i^j is composed of a link $r_i^j.\varepsilon$, and the queue number $r_i^j.q$ of the link origin. For example, the second link assignment in the route r_3 is denoted with r_3^2, the associated link $r_3^2.\varepsilon$ is $\varepsilon_{5,8}$, and the associated queue number $r_3^2.q$ is 1 which indicated that the first queue of the port $\varepsilon_{5,8}$ is used.

Each route starts with a link assignment originating from the talker $s_i.v_s$, and ends

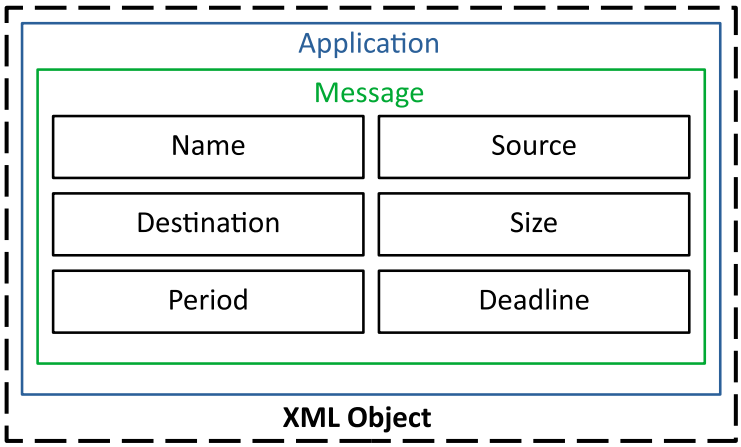

Figure 3: Application Input Structure

with a link to the listener $s_i.v_d$. The number of link assignments in the route r_i is denoted with $|r_i|$, and it starts from 2 since we assume there is at least one switch in the route.

Input Format: The applications are given as an XML file with the tags denoting the application model. The structure of the XML file and the corresponding tags are shown in Fig. 3. The relevant notation to the tags are presented in Table 1.

3 Problem Formulation

We formulate the problem as follows: Given (1) the set of all flows $\mathcal{S}$ in the system, (2) the network graph $\mathcal{G}$, we are interested in synthesizing the routing tables, the cycle tables, and the queue tables in the network such that (a) all the flows in the system are schedulable (their deadlines are satisfied) and (b) the mean bandwidth utilization of all links, as defined in Section 4.3, is minimized. Synthesizing the routing tables, the cycle tables, and the queue tables is equivalent to determining the set of routes $\mathcal{R}$.

4 Constraints and Objective

In this section we present the formalism governing a CSQF network which is helpful for implementing solutions. Although the solution for this problem can be implemented with any methods such as heuristics and Constraint Programming (CP), the formalism does not change. There are only two constraints that govern a CSQF network which are presented in Section 4.1 and Section 4.2. We also present the objective function for the optimization problem in Section 4.3.

Cycle Domain Transformation: Since CSQF networks operate based on same length cycles, we need to transform from time domain to cycle domain to define the governing constraints. To this end and for the architecture model, we define the capacity S which is equivalent to the bandwidth s of a link , and the induced delay D which is equivalent to the propagation delay function d of a link. The capacity S specifies the data size in Bytes which the link transfers in a cycle and is calculated using $S = s \times |c|$.

The induced delay D specifies the worst-case total delay in cycles and is calculated using $D = d/|c|$.

Similarly, we define the arrival pattern $\mathcal{A}$ which is equivalent to the size and period of a flow, and acceptable delay $\mathcal{D}$ which is equivalent to the deadline d of a flow. The arrival pattern specifies the data in Bytes that is sent from the talker in each cycle and is calculated using Eq. (1). The acceptable delay $\mathcal{D}$ specifies the maximum allowed end-to-end delay in cycles and is calculated using $d/|c|$.

$$s_i.\mathcal{A}(c) = \begin{cases} s_i.c & \text{for} \quad c \times |c| \% s_i.t = 0 \\ 0 & \text{for} \quad c \times |c| \% s_i.t \neq 0 \end{cases} \tag{1}$$

4.1 Deadline constraint

This constraint imposes the restriction that all flows in the application model must meet their deadlines. Assuming that each flow s_i will be transmitted via the route r_i, each link assignment r_i^j carries link delay $r_i^j.\varepsilon.D$ and the associated queue number $r_i^j.q$ which shows the number of delay cycles as described in Section 2.2. With the cycle shift and delay for each link assignment in r_i, the end-to-end delay of the flow is determined.

This constraint iterates over all flows, determines the end-to-end delay of each flows (denoted with $E2E$) and checks for the delay not to exceed the flow deadline $s_i.\mathcal{D}$, all defined in cycle domain. The end-to-end delay is the sum of induced delay (how many cycles it takes to propagate the flow bits along the link medium) and the cycle shift (how many cycles the flow is delayed in the source node of the link). The constraint is formally defined in Eq. (2).

$$\begin{aligned} &\forall s_i \in \mathcal{S} : \\ &E2E = \sum_{j=1}^{|r_i|} (r_i^j.\varepsilon.D + r_i^j.q) \\ &E2E \leq s_i.\mathcal{D} \end{aligned} \tag{2}$$

4.2 Link capacity constraint

This constraint enforces solutions to meet the link capacity limit. Each link in the architecture must only transmit less amount of data than its capacity in each cycle. For each link. the constraint is imposed on the flows that are transmitted via routes that include the link. Furthermore, the arrival function of these flows should be shifted to the time when they are received by the link. The constraint is defined in Eq. (3).

This constraint computes the consumed bandwidth of each link $\varepsilon_{i,j}$ in each cycle c, which is denoted with $B_{i,j}^c$ and checks for the consumed bandwidth not to exceed the link capacity $\varepsilon_{i,j}.S$, all in cycle domain. The consumed bandwidth in each cycle $B_{i,j}^c$ is defined as the sum of sizes of flows that are passing through the link in the cycle. To this end, first we find the flows that are using the link for transmission and then determine the latency α that takes for each flow to leave the source node of the link. We use the arrival pattern function $\mathcal{A}(c)$ and the latency α for shifting the pattern to find the flow size at the cycle. The objective function is defined in Eq. (4).

$$
\begin{aligned}
&\forall c \in C, \forall \varepsilon_{i,j} \in \mathcal{E}, \\
&\forall s_k \in \{\mathcal{S} | r_k^l.\varepsilon == \varepsilon_{i,j}\}, \\
&\alpha = \sum_{m=1}^{l-1} (r_k^m.\varepsilon.D + r_k^m.q) : \\
&B_{i,j}^c = \sum s_k.\mathcal{A}(c-\alpha) \\
&B_{i,j}^c \leq \varepsilon_{i,j}.S
\end{aligned}
\tag{3}
$$

4.3 Objective Function

The objective in this problem is to minimize the mean bandwidth utilization of all flows subject to the deadline and link capacity constraints. The function uses the parameter $B_{i,j}^c$ defined in Eq. (3) which represents the bandwidth utilization of the link $\varepsilon i, j$ in cycle c.

This function uses the consumed bandwidth of each link $B_{i,j}$ and computes the mean bandwidth usage of all links Ω. The consumed bandwidth is the equal to the maximum bandwidth usage in each cycle.

$$
\begin{aligned}
&\forall \varepsilon_{i,j} \in \mathcal{E}, \forall c \in C : \\
&B_{i,j} = \frac{Max(B_{i,j}^c)}{\varepsilon_{i,j}.S} \times 1000 \\
&\Omega = \frac{\sum B_{i,j}}{|\mathcal{E}|}
\end{aligned}
\tag{4}
$$

5 Evaluation

We evaluate the performance of the solution on several Test Cases (TCs). We have defined total number of 9 synthetic TCs which are split into three groups: small TCs, medium TCs, and large TCs. All TCs follow the same input format as given in Section 2.

Output format: We expect that the solution generates an XML output files. The generated output file should contain the following information. Fig. 4 presents the structure of the output file.

- Runtime for generating the solution
- Objective value of the solution
- Mean E2E delay of the solution
- Maximum E2E delay for each flow
- Routing for all flows

The runtime for generating the solution is in seconds and represents the time takes for generating the solution. The objective value of the solution is the value of the function Ω in Eq. (4).

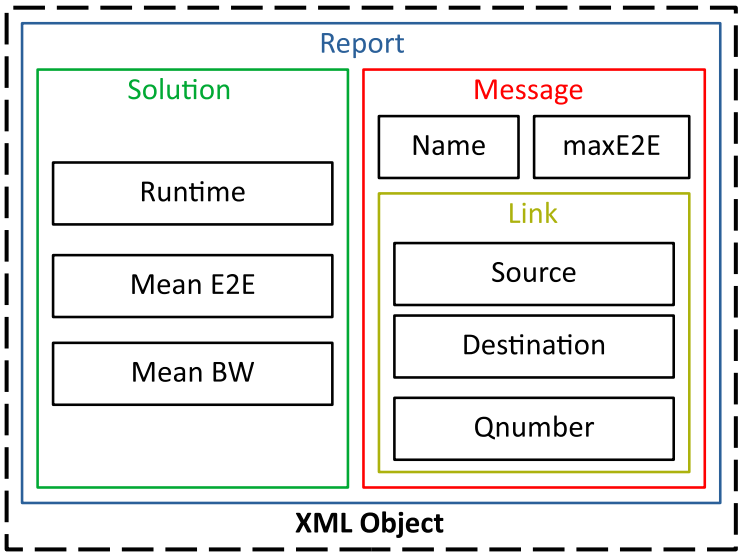

Figure 4: Report Output Structure

The mean E2E delay of the solution shows the mean E2E delay of all flows, and it is in cycles. To calculate the mean E2E delay of the solution, one should first calculate the mean E2E delay of each flow and then calculate the mean value of these delays. For example, the flow s_1 has E2E delay of 4 cycles. Assuming 6 cycles for the mean E2E delay of the flow s_2, the mean E2E delay of the solution composed of s_1 and s_2, is 5 cycles.

The maximum E2E delay for each flow is given in cycles. The routing for all flows is also reported separately.

References

[1] Aeronautical Radio, Inc. *ARINC 664P7: Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network*. 2009.

[2] Mach Chen, Xuesong Geng, and Zhenqiang Li. Segment routing (SR) based bounded latency. *Internet Engineering Task Force, Internet-Draft draft-chendetnet-sr-based-bounded-latency-00*, 2018. Available at `https://datatracker.ietf.org/doc/html/draft-chen-detnet-sr-based-bounded-latency-01`.

[3] J. D. Decotignie. Ethernet-based real-time and industrial communications. *Proceedings of the IEEE*, 93(6):1102–1117, 2005.

[4] Joachim Feld. Profinet—scalable factory communication for all applications. In *Factory Communication Systems, 2004. Proceedings. 2004 IEEE International Workshop on*, pages 33–38. IEEE, 2004.

[5] IEEE. *802.3 Standard for Ethernet*, 2015.

[6] IEEE. Official Website of the 802.1 Time-Sensitive Networking Task Group. `http://www.ieee802.org/1/pages/tsn.html`, 2016.

[7] Dirk Jansen and Holger Buttner. Real-time ethernet: the ethercat solution. *Computing and Control Engineering*, 15(1):16–21, 2004.

[8] AS Oliver Kleineberg and Axel Schneider. Time-sensitive networking for dummies, 2018.

[9] John Rushby. Bus architectures for safety-critical embedded systems. In *Embedded Software*, pages 306–323. Springer, 2001.

[10] SAE. *AS6802: Time-Triggered Ethernet*. SAE International, 2011.