FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Performance Evaluation and Optimization of Standard-Ethernet Traffic in TTEthernet Systems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor/in der technischen Wissenschaften

by

Ekarin Suethanuwong
Registration Number 0628110

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: O. Univ. -Prof. Dr. Hermann Kopetz

The dissertation has been reviewed by:

(O. Univ. -Prof. Dr. Hermann Kopetz) (Prof. Dr.-Ing. habil. Roman Obermaisser)

Wien, 10 November 2011

(Ekarin Suethanuwong)

Erklärung zur Verfassung der Arbeit

Ekarin Suethanuwong
Brigittenauer Länder 222-224, 1200 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 10 November 2011	
(Ort, Datum)	(Unterschrift Verfasser)

Performance Evaluation and Optimization of standard-Ethernet Traffic in TTEthernet Systems

Nowadays standard Ethernet is widely used in home and office computer networks, and well recognized as the low-layer protocol of the Internet protocol. There are three main advantages of using today's Ethernet technology: (1) high bandwidths, for example with Fast Ethernet and Gigabit Ethernet, (2) open standard protocol, defined in IEEE 802.3, and (3) low prices of commercial-off-the-shelf standard-Ethernet devices. In additional, today's Ethernet technology, which is so-called switched Ethernet, has higher bandwidth utilization than traditional Ethernet based on bus network topology, since switched Ethernet provides full-duplex communication and no collision occurrence. However standard-Ethernet, which was not originally designed for real-time communication, does not provide a timely-determinism property. In the last decade there have been many designs that adapt standard Ethernet to provide temporal guarantees. Such a standard-Ethernet based protocol augmenting with temporal guarantees is called real-time Ethernet (e.g. Ethernet POWERLINK, SERCOS III, PROFINET IRT, Ethernet/IP, MODBUS-TCP, AFDX, and TTEthernet).

Recently there has been a main driving force behind applications of real-time Ethernet that is the increasing demand for a seamless connectivity between legacy Ethernet networks and real-time Ethernet networks. This can be seen in the industrial automation domain, for example, the connectivity between the administrative tasks of a company (office networks) and the factory floor (fieldbus networks) has been increased. Another main driving force for adapting standard Ethernet to real-time Ethernet is to integrate all application types into a single standard-Ethernet based network. Such a single standard-Ethernet based network is potentially very cost effective, and reduces the complexity of network infrastructure.

TTEthernet was designed to be a novel unified communication architecture that meets the requirements of all application types from non real-time applications, to real-time applications, to the most demanding safety-critical real-time applications. TTEthernet were aimed to altogether achieve the three main properties: (1) determinism property for real-time traffic, (2) full standard-Ethernet compatibility for real-time communication, and (3) high Ethernet-bandwidth utilization for standard-Ethernet traffic. Due to the fact that TT traffic in TTEthernet systems takes precedence over standard-Ethernet traffic, the flow of standard-Ethernet traffic depends only on a TT-traffic communication schedule. The objective of this thesis is to investigate a TT-traffic scheduling approach for obtaining the highest performance of standard-Ethernet traffic coexisting with the TT traffic in the same TTEthernet systems. The throughput and timing-performance of standard-Ethernet traffic are evaluated in both simulated and physical 100-Mbps TTEthernet systems. For the simulated TTEthernet systems a simulation model for TTEthernet systems is developed and implemented.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Ethernet technology is well established and widely used in home and office networks, known as Local Area Networks (LANs). The benefits of using Ethernet networks include: the low cost of commercial-off-the-shelf Ethernet devices, and the extremely high bandwidths available; for example with Fast Ethernet and Gigabit Ethernet. Additional benefits include that Ethernet is an open standard, defined in IEEE 802.3 [IEE05a]. Today's Ethernet technology is based on switched Ethernet [IEE04], where the Ethernet controllers of computer nodes are interconnected via switching devices called Ethernet switches. Switched Ethernet eliminates the disadvantages of traditional Ethernet based on a bus topology; i.e. switched Ethernet has full-duplex communication and no collision occurrence. Furthermore, advanced features of switched Ethernet enhance Ethernet utilization, e.g. virtual LAN [IEE05b]. Ethernet technology used in *open world systems* [KAGS05] such as the Internet has therefore become desirable for adoption in distributed systems of *closed-world systems* [KAGS05]; such as industrial automation networks, aerospace networks, and automotive networks. These distributed systems of closed-world systems require temporal guarantees for message communication. The fact that switched Ethernet was not originally designed for real-time communication is therefore problematic. With advancements in modern Ethernet technology however, switched Ethernet can now obtain real-time communication guarantees under certain conditions; i.e. very light traffic loads [LL02] [LL06a] or clearly separated routes of Ethernet frames in Ethernet switches [Ana02]. However, it would be futile for distributed real-time systems to be unable to utilize the high bandwidths available from standard Ethernet. In the past decade there have been many designs [Fel05] [Dec05] that have adapted standard Ethernet to be capable of: providing temporal guarantees, and obtaining a better utilization of today's Ethernet bandwidths for real-time communication. Such a standard Ethernet-based protocol augmenting with temporal guarantees is called *real-time Ethernet* [Fel05] [WK06].

Besides achievable real-time performance of real-time Ethernets, one of the main driving forces behind applications of real-time Ethernets is the increasing demand for a seamless connectivity between Ethernet networks in *open world systems* and real-time Ethernet networks in *closed-world systems*. For example, in the industrial automation domain [JISW09], the demand for connecting between the administrative tasks of a company (office networks) and the factory floor (fieldbus networks) has increased. Another interesting point of adapting standard Ethernet for use in *closed-world systems* is that integrating all types of applications into a single network based on standard Ethernet is potentially very cost effective. One example from the automotive domain shows that the Ethernet-based unified communication network infrastructure, which is used for all application types in a car including: infotainments, multimedia, and engine control applications, reduced both the *complexity and cost* of such typical network infrastructures [MEM08]. Another outstanding protocol of real-time Ethernet for integrating all application types into a single network infrastructure is TTEthernet [TTE08]. TTEthernet was designed to be a novel unified communication architecture, based on Ethernet, that meets the requirements of all applications types; from non real-time applications (e.g. web browsers), to real-time applications (e.g. multimedia applications), to the most demanding safety-critical real-time applications (e.g. fly-by-wire applications). Therefore, TTEthernet is inherently suited for mixed-criticality systems [SB08] [SBH09].

1.1 Problem Statement

In real-time Ethernet networks, messages can be classified into two message classes: *non real-time messages* and *real-time message*s [Ste06]. The non real-time messages are standard Ethernet messages that are transported according to the standard Ethernet specification [IEE04]. These non real-time messages can be seen in standard Ethernet networks, e.g. office or Internet networks. The real-time messages are standard Ethernet-based messages that are transmitted with temporal guarantees; i.e. transmission latency and jitter are bounded within their specified values. It is imperative that non real-time messages do not interfere with the timely behavior of real-time messages, which coexist in the same real-time Ethernet network.

1.1.1 Dataflow integration in a real-time Ethernet network

Dataflow integration in a real-time Ethernet network denotes that both non real-time traffic (standard Ethernet traffic) and real-time traffic can coexist in the same physical network. Most real-time Ethernet networks are capable of dataflow integration. We classify dataflow integration according to the network topology of real-time Ethernets: line or ring topology, or star topology. These are described as follows:

- *Line or ring topology*: Some protocols of real-time Ethernets are suitable for a line or ring topology, e.g. Ethernet POWERLINK [EPS08] and SERCOS III [SER07]. These protocols allow non real-time traffic to flow in the network if and only if all devices are in a synchronized state. In an Ethernet POWERLINK network, only one device of the controlled nodes is granted permission from the managing node to send non real-time messages in the asynchronous phase. Therefore, the bandwidth utilization for non real-time traffic in an Ethernet POWERLINK network is low. For SERCOS III, the protocol uses a logical ring network structure, and devices can only transmit non real-time messages in a certain interval (called the non real-time channel) of a communication cycle. The non real-time channel in a SERCOS III device is established after the communication cycle is synchronized. In other words, the non real-time messages in a SERCOS III device are delayed until the non real-time channel of the communication cycle is already established. This makes SERCOS III unable to fully utilize Ethernet bandwidths for non real-time traffic.

- *Star topology*: Some real-time Ethernets were designed to be suitable for a star network topology; e.g. PROFINET IRT [PI09], Ethernet/IP [Pau01], MODBUS-TCP [Acr05], AFDX [Con05], and TTEthernet [TTE08]. Each of these protocols is further described in *Chapter 3 - State of the art*. These protocols do not change the network structure of modern Ethernet technology, *switched Ethernet* [LL02]. In a star topology, switching devices are responsible for handling both non real-time traffic and real-time traffic. The switching devices aim to maintain all functionalities of standard Ethernet switches for the transmission of standard Ethernet traffic (the store and forward mechanism), as well as adding a mechanism in order to provide temporal guarantees for real-time traffic. Therefore, these real-time Ethernets allow legacy Ethernet devices to connect to the networks without modification. This is not the case for AFDX however, which uses a bandwidth limiting approach whereby a legacy Ethernet device cannot directly connect to an AFDX Switch. This is because there is no mechanism to handle the non real-time messages interfering with the timely behavior of the real-time messages. In other real-time Ethernets based on a star topology however, there are mechanisms that handle both real-time traffic and non real-time traffic, so that the non real-time messages cannot interfere with the timely behavior of real-time messages (in the same real-time Ethernet device). With these protocols, non real-time messages from legacy Ethernet devices can be arbitrarily sent into the connected switching device. These non real-time messages can be stored in the buffer of the switching device, and then sent out if there are no real-time messages currently transmitting on the same output port. The bandwidth utilization for non real-time messages in these real-time Ethernet networks is higher than the ones in real-time Ethernet networks using a line or ring topology. Therefore, we aim to focus the dataflow integration in real-time Ethernet networks based on a star topology. Based on our literature review we classify the dataflow integration mechanisms of a switching device, in real-time Ethernet networks based on a star topology, into three main approaches: Message Prioritization, Two-channel Partition, and Time-triggered Communication Scheduling.

  1. *Message Prioritization approach*: This approach utilizes the *tag* field of an Ethernet frame, as defined in IEEE 802.1Q [IEE05b]. The *tag* value is used to identify certain types of traffic. Real-time Ethernets using this approach (e.g. Ethernet/IP and MODBUS-TCP) make use of standard, commercial-off-the-shelf Ethernet switches as switching devices. A real-time message is prioritized over non real-time messages by specifying the higher-priority traffic class in the *tag* field of the real-time message. An Ethernet switch queues traffic of the same class on a first-come, first-served basis. A real-time message in the Ethernet switch cannot preempt a non real-

time message that is currently transmitting. Therefore, such real-time messages will be delayed until the non real-time message has been completely transmitted. With this approach, more than one real-time message with the same specified priority could be simultaneously received at a switching device. Consequently, some of the real-time messages may be delayed, and in turn miss their deadlines. This approach is recommended for use in networks where the amount of traffic is limited. Such limiting can be achieved using the recommendations for Ethernet/IP [Ana02], or bandwidth limiting mechanisms (e.g. the Token-bucket approach [ARI06]) for AFDX. Using such methods, the timely behavior of all real-time messages can be maintained.

2. *Two-channel Partition approach*: This approach is used in PROFINET IRT. With this approach a communication cycle is partitioned into two channels: a *real-time channel* and a *non real-time channel*. The real-time messages can be transmitted only in the real-time channel, whereas the non real-time messages can be transmitted only in the non real-time channel. In a switching device, the non real-time messages have to wait until the time interval of the non real-time channel arrives. Additionally, the non real-time messages can only be forwarded if the switching device is in the synchronization state. Otherwise, the non real-time messages remain in the buffer of the switching device. This approach has substantial advantages over the *Message Prioritization* approach, namely that the non real-time messages cannot interfere with the timely behavior of the real-time messages in a switching device. Furthermore, the real-time traffic in this approach can utilize high Ethernet bandwidth without a quantity limitation on the real-time traffic. The major drawback of this approach is that non real-time messages cannot pass through a switching device if synchronization in the switching device is not established.

3. *Time-triggered Communication Scheduling approach*: This approach is used in TTEthernet [TTE08], which supports mixed-criticality applications [Mir09] [SMJ08]. With this approach, the transmission of real-time messages is performed according to a predefined, time-triggered communication schedule (TT schedule). A switching device called a *TTE switch* handles the real-time messages according to the TT schedule, and the non real-time messages according to the standard Ethernet specification (i.e. *store and forward*). All the time-triggered Ethernet devices (TTE switches and TTEthernet end systems) in a TTEthernet network contain a common TT schedule. The real-time messages can pass through a TTE switch if two conditions are satisfied: the TTE switch is in the synchronization state, and the real-time messages arrive in the duration of the specified acceptance window [TTE08]. TTEthernet End systems that are in the synchronization state enable the transmission of real-time messages with respect to the predefined TT schedule. A real-time message is called a time-triggered message, and a non real-time message is called an event-triggered message. According to [SBH09], there are three integration methods

to solve the confliction between time-triggered and event-triggered messages in a TTE switch. These are described as follows:

- *Preemption* method: An event-triggered message transmitting at a TTE switch is preempted, as soon as a time-triggered message arrives. The event-triggered message will therefore have to be re-transmitted as soon as possible.
- *Timely Block* method: With a predefined TT schedule, a TTE switch can relay an event-triggered message, if it is guaranteed that the transmission of the event-triggered message will have completed before a time-triggered message has to be relayed. Otherwise, the event-triggered message is blocked in the buffer of the TTE switch, so that the time-triggered message can be timely relayed.
- *Shuffling* method: An event-triggered message continues to be relayed by the TTE switch when a time-triggered message arrives. The time-triggered message will be transmitted after the relay process of the event-triggered message is completed.

1.1.2 Performance of standard Ethernet in a TTEthernet network

Besides the determinism property of real-time traffic in TTEthernet, another outstanding feature of TTEthernet is that of high bandwidth utilization for the non real-time, standard Ethernet traffic. This high utilization is possible because non real-time messages in a TTE network can pass through a TTE device irrespectively of synchronization state. The flow of non real-time traffic in a TTE device can therefore occur during either the non-synchronization state, or the synchronization state. In the non-synchronization state, the non real-time messages can pass through an unsynchronized TTE device according to the standard Ethernet specification [IEE04]. In the synchronization state, non real-time messages can only pass through the synchronized TTE device when there are no time-triggered messages requiring transmission (according to the time-triggered communication schedule). If a real-time message is relaying at a TTE device, the non real-time messages are delayed, remaining in the device's buffer. In other words, non real-time messages cannot interfere with the timely behavior of the real-time messages. The performance of non real-time traffic in a TTE network is strongly dependent on the scheduling of time-triggered traffic. In this thesis we propose a new scheduling approach for the time-triggered traffic in a TTEthernet network. The purpose of this scheduling approach is to optimize the performance of standard Ethernet traffic. We present a mathematical analysis of each of the possible TT schedules obtained during the induction of our scheduling approach. We then report the most optimal TT schedule. In addition, we propose a TTEthernet simulation model based on discrete event simulation. The behavior of switched Ethernet in this simulation model is calibrated using existing results from published documents. We prove our scheduling approach using both the simulation model and experiments on a real TTE system. The results show that the full performance of standard Ethernet is possible in a TTEthernet system using TT schedules from our scheduling approach.

1.2 Structure of Thesis

The remainder of this thesis is organized as follows:

- Chapter 2 gives an overview of the basic terms and concepts of distributed real-time systems. The two paradigms of distributed communications: time-triggered and event-triggered, are briefly described, as is the determinism property in real-time communication. This chapter also describes the terms of open and closed world systems. Lastly, the concept of discrete event simulation is addressed, upon which our proposed simulation model is based.

- Chapter 3 presents the state of the art of *real-time Ethernet*. Real-time Ethernets from the industrial automation domain (e.g. Ethernet/IP, SERCOS III, and PROFINET) and the aerospace domain (e.g. AFDX) are described, as is TTEthernet.

- Chapter 4 presents our TTEthernet simulation model. Firstly the relationship between simulation time and local clock time is described. All the system components of a TTEthernet system are then explained. The calibration results from simulated TTEthernet systems based on our simulation model with the results from published papers are shown.

- Chapter 5 proposes our TT-traffic scheduling approach for TTEthernet systems. With this approach, the possible time-triggered communication schedules of given time-critical applications are obtained; i.e. periods and offsets of the time-triggered traffic from the given time-critical applications. The mathematical analysis of this approach is expressed to show the bandwidth utilization of the standard Ethernet traffic for each obtained schedule. Two possible forms of offsets in the schedules are described: *continuous offset form* and *distributed offset form*. Examples of time-critical applications are given. The optimized schedule for these time-critical applications is then obtained using our scheduling approach.

- Chapter 6 presents and analyzes the results of Ethernet performance in a TTEthernet system using our proposed scheduling approach. The specified periods of given time-critical applications are used in our TT-traffic scheduling approach. The experimental setups of TTEthernet system are fully described. This includes: the network structure of TTEthernet systems, how to generate the time-triggered messages from a TTEthernet end system regarding the obtained schedules, and how to generate standard Ethernet messages from commercial-off-the-shelf Ethernet devices.

- Chapter 7 presents the conclusion of this thesis, and gives an outlook for future work.

CHAPTER 2

Basic Concepts

This chapter presents the basic concepts and terminology used in this thesis. The first section begins with an overview of *distributed real-time systems,* and the classification of communication systems with regard to communication trigger. The second section focuses on the basic concept of *time-triggered communication,* a fundamental part of TTEthernet. It comprises the following relevant topics: reference clock, local clock, global time base, clock synchronization, cluster cycle, message permanence, and determinism. The third section, *Ethernet technology,* describes the history and operational principle of shared and switched Ethernet. In the fourth section we present the *performance metrics of switched Ethernet,* which are used to quantify channel utilization (throughput) and timing behavior (latency). We also provide an overview of *mixed-criticality systems* in a TTEthernet network. This chapter ends with the concept of *discrete event simulation,* used in building our simulation model for TTEthernet systems.

2.1 Distributed Real-Time Systems

A real-time computer system is a computer system in which correctness of the system behavior depends not only on the logical results of the computations, but also the physical instant at which these results are produced [Kop97, pp.2]. A real-time computer system and its environment form the *real-time system* depicted in Fig. 2.1. A real-time system can be decomposed into the following three sub-systems (called clusters): *operator cluster, computational cluster,* and *controlled cluster* [Kop97]. The operator cluster and controlled cluster form the environment of a real-time system, whereas the computation cluster refers to the real-time computer system. Humans can deal with a real-time computer system at the operational cluster via input devices (e.g. keyboard or push button) and output devices (e.g. display panel or touch screen). The controlled objects in the controlled cluster are interfaced with the operational cluster through sensors (e.g. thermocouple, or tachometer) and actuators (e.g. motor or electrical heater). The real-time computer system must react to stimuli from the environment (controlled objects and operators) within time intervals specified via *deadlines*. A deadline is a bounded instant by which a real-time computer system must react in order to satisfy the required timing behavior. With regard to the consequence of missing a specified deadline, real-time computer systems are classified into two categories: *hard real-time computer systems* and *soft real-time* *computer systems*. If the consequence of missing a deadline may result in catastrophe (e.g. human injury), such a real-time computer system is called a hard real-time computer system. A hard real-time system is also called a *safety-critical real-time computer system* [Kop97]. If the consequence of missing a deadline will not result in a catastrophe, only degrade system performance, such a real-time computer system is called a soft real-time computer system.

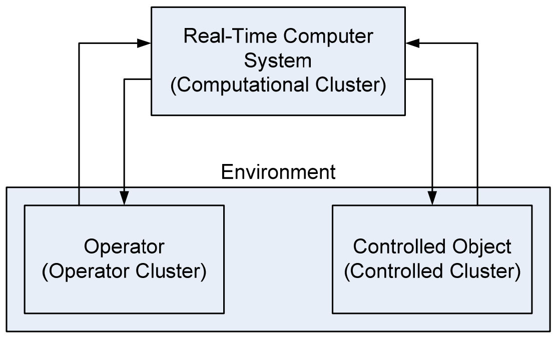

Real-Time Computer System (Computational Cluster)
Environment
Operator (Operator Cluster)
Controlled Object (Controlled Cluster)

Fig. 2.1: A real-time system

A d*istributed real-time system* is a real-time system that consists of a set of real-time computer systems (called nodes) that are distributed and interconnected via a *real-time communication system*. Each node in a distributed real-time system can be divided into two sub-systems: a *communication controller* and a *host computer*. The interface between the host computer and the communication controller is called the *communication network interface* (CNI). The set of communication controllers and the common communication network form the *communication system* [Obe05]. The communication system is used for transporting messages from the CNI of a sender node to the respective CNI of the receiver nodes. A real-time communication system is a communication system in which the correctness of message communication does not only depend on valid message content, but also on the timely delivery of messages.

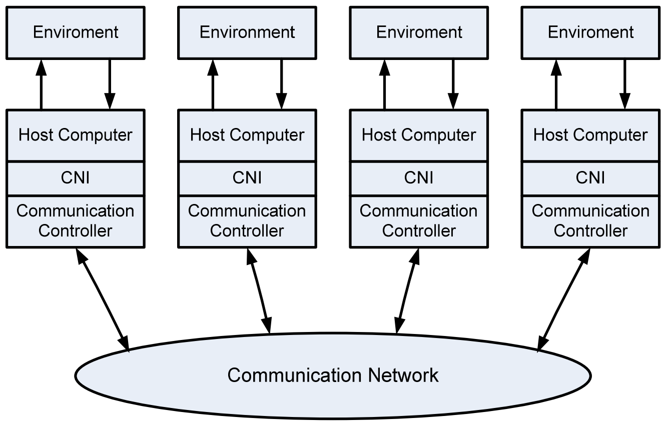


Fig. 2.2: A distributed real-time system

2.1.1 Characteristics of a communication channel

The characteristics of a communication channel are determined by the following two properties: *bandwidth* and *propagation delay*.

- *Bandwidth* is the maximum number of message bits that can be transmitted in a communication channel in a single unit of time. For example, the bandwidths of Fast Ethernet and Gigabit Ethernet are 100 Mbps and 1000 Mbps, respectively. Bandwidth is a physical characteristic of a channel, and thus remains constant over time.

- *Propagation delay* is the time interval that a message bit takes to get from one end of a communication medium to the other. The propagation delay of a copper cable is approximately 2/3 of the transmission speed of light in vacuum (i.e. 2/3 * 300,000 km/second).

2.1.2 Event and State Messages

There are two types of message information: *event information* and *state information* [Kop97]. Event information is the value of an event (a *significant change* in the value of a state variable), whereas state information is the current value of a state variable (e.g. temperature). Message semantics related to event information and state information are referred to as *event messages* and *state messages* [Kop97].

- *Event messages* contain event information sent from a sender node *exactly once* after *the occurrence of an event*. When event messages have arrived at a receiver node, they are queued and then read sequentially. Each message is discarded after it is read. This denotes that the data in the queue is consumed by the reading process of the host computer at the receiver node.

- *State message*s carry state information. The communication controller of a sender decides autonomously to send a state message, i.e. it is independent of event occurrences or any control signal from the host computer of the sender. The data in a state message is always stored in the same place at the CNI of the sender node. Since a *state variable* is valid for a time period, the host computer of the sender node can replace the old state data with the new data. The communication controller reads the data periodically from the CNI of the sender node, and then sends a state message with that data. The data stored at the CNI is not consumed by the sending process. The state message data that arrives at the receiver node overwrites the old data. The reading process of the host computer at the receiver node does not remove the data from the CNI, i.e. the data is not consumed by the reading process of the host computer.

2.1.3 Event-triggered and Time-triggered Communication Systems

A communication trigger is an event that initiates the transmission of a message in the communication system of a distributed real-time system. Communication triggers are classified as either: event-triggered or time-triggered [Kop97]. The difference between these triggers depends on the *sources of control signals*: event-triggered control signals are derived from a *significant change of state* in the environment of a host computer, whereas time-triggered control signals are derived from a *predefined instant* in a communication system. Communication systems are classified according to their communication triggers as either: *time-triggered communication systems* or *event-triggered communication systems* [Kop97] [Obe05]. These are described as follows:

➢ ***Event-triggered communication systems*:** The control signals of an *event-triggered* communication system are external to that system. The decision of when to send a message in an event-triggered communication system is made by the application software in a host computer. Transmission of all messages in an event-triggered communication system is initiated whenever the CNIs receive control signals (interrupt mechanisms) from the host computers. Messages that are generated by control signals from event triggers are called *event-triggered messages*. An example of a well-known communication protocol that transports event-triggered messages is *Ethernet*. In an event-triggered communication system, more than one node can simultaneously transmit their respective messages to a particular receiving node. In a shared Ethernet network, the *Carrier Sense Multiple Access/Collision Detection (CSMA/CD)* mechanism resolves the access conflict from the concurrently sending messages. However, this causes uncertain network latencies and low network utilization. In a switched Ethernet network, the concurrently sending messages will be queued in an Ethernet switch. Therefore, some of these messages may be delayed, or dropped out of the switch's buffer (this is known as a *buffer overflow*).

➢ ***Time-triggered communication systems*:** The control signals of a *time-triggered* communication system reside within that system. Thus the decision of when to send a message is taken by the system. All communication controllers in a time-triggered communication system must have consistent views of global time, and contain the global communication schedule. All the messages in a time-triggered communication system are initiated by time-triggers, i.e. the control signals are generated when the progression of global time reaches the predefined instant specified in the global communication schedule. In a time-triggered communication system, the host computers cannot influence the temporal behavior, i.e. no control signals from the host computers are able to cross the CNIs. It is not possible for control signal errors to propagate from the host computers to the communication system, and vice versa [Kop97]. The messages that are activated by time-triggers are called *time-triggered messages*. An example of a well-known protocol that transmits time-triggered messages is the *Time-Triggered Protocol* (TTP) [KG93].

2.2 The concepts of time-triggered communication for TTEthernet

TTEthernet is a communication protocol that adopted the time-triggered communication paradigm into Ethernet networks [KAGS05]. In this section we present the basic concepts of time-triggered communication [Kop97], as these are a prerequisite for understanding TTEthernet. We describe the following concepts of time-triggered communication:

2.2.1 Reference Clock and Local Clock

In a time-triggered communication system, each communication controller has an individual *local clock* that is used for capturing the progression of time. The time that is indicated by the local clock of a node is called *local clock time*. Note that the communication controller and the host computer of a node can have either the same local clock or individual local clocks. In this thesis we consider the host computer and communication controller of a node to share the same local clock. The local clock of a node contains a *physical clock* that consists of a counter, and a physical oscillator mechanism. The physical oscillator (e.g. crystal resonator) increases the counter value every constant time interval. Each time event in which the counter value is increased by a physical oscillator is called a *microtick*. The granularity of a local clock's time is a set of microticks, and is referred to as a *macrotick,* or simply, *tick*.

In abstraction, a *reference clock* is defined as a clock that operates with a very high frequency, and is in perfect agreement with the international standard of time, known as the International Atomic Time (TAI) [Kop97]. The TAI officially defined the measure standard of a second based on the *atomic clock* (cesium-133 atom). *TAI is a chronoscopic timescale, i.e. a timescale without any discontinuities* [Kop97, pp. 51]. Therefore, in a time-triggered communication system, we refer to the time defined by TAI.

Clock drift is the drift of a physical clock *k* between microtick *i* and microtick *i+1*. It denotes the frequency ratio between this clock *k* and the reference clock, at the instant of microtick *i*. The drift is determined by measuring the duration of a granule of the clock *k* with the reference clock and dividing it by the nominal number of reference clock microticks in a granule [Kop97]. If the value of a clock drift is equal to one, then the physical clock is running at the same speed as the reference clock. If the clock drift's value is greater than one, the physical clock is running faster than the reference clock.

Drift rate is a practical term that expresses the *clock drift* of a physical clock. The definition of drift rate is the absolute value of the subtraction between the clock drift and 1, as expressed in Equation 2.1. A physical clock which has a drift rate of zero is called the *perfect clock.*

Drift rate = |clock drift -1|……………………………………………………………..(2.1)

In reality, physical clocks are not perfect clocks. This is due to varying drift rates caused by the influence of environmental conditions, e.g. a change of the ambient temperature, a change of the voltage level supplied into a resonator, or the aging of a resonator [Kop97]. The *maximum drift rate* of a resonator in a physical clock is guaranteed, and specified in the data sheet from the manufacturer. Typical maximum drift rates of resonators are in the range of 10^{-2} to 10^{-7} sec/sec [Kop97].

The *drift offset* of two physical clocks is the time difference between their respective microticks. The *precisio*n of all physical clocks in a time-triggered communication system denotes the maximum drift offset between any two physical clocks during a specific interval of interest.

2.2.2 Global Time and Clock Synchronization

For the operation of a time-triggered communication system, the global-time base is of significant importance. This is because each communication controller in a time-triggered communication system transmits its time-triggered messages with respect to the common communication schedule. A prerequisite to the transmission of time-triggered messages is that the system-wide global-time base at each communication controller must have already been established via a clock synchronization mechanism. The local view of the global time of a node is indicated by its local clock time.

Due to the fact that the drift rates of any two physical clocks will not be identical, any two local clocks will always run apart. The clock synchronization process in a time-triggered communication system will bring the local clock times of all nodes into close relation with each other (i.e. within a specified precision). Each node must periodically synchronize its local clock with the local clocks of all other nodes in order to establish a global time base with a specified (bounded) precision. The resynchronization period is called the *resynchronization interval* (R_{int}). The (possible) maximum drift offset of any two physical clocks in a time-triggered communication system during the resynchronization interval *is not greater than* $2\rho R_{int}$, where ρ is the maximum specified drift rate of the two physical clocks.

In a time-triggered communication system, there is always the *jitter* of communication delay; the difference between the maximum and minimum transport delays of all messages. Jitter affects the transport of a message between two nodes. Therefore, the achievable precision of the clock synchronization does not only depend on the drift offset between any two physical clocks, but also on the jitter of communication delay. Thus, the *clock error* of the clock synchronization between two nodes is the summation of the clock offset between the two nodes and the deviation of the transport delay from the expected one.

2.2.3 Cluster Cycle

A *real-time cluster* is a set of cooperating computing components that are connected by a real-time communication system [Kop08]. A time-triggered communication system sends and receives messages according to the (common) communication schedule, which is derived from scheduling time-triggered traffic into a cluster cycle. A cluster cycle is an interval of time containing events of time-triggered message transmissions. The length of a cluster cycle is typically designed to be equal to the Least Common Multiple (LCM) of all periods of real-time applications in a real-time cluster. All transmission instants of real-time applications in a real-time cluster are assigned in a cluster cycle using two terms: *period* and *offset*. The offset in a cluster cycle denotes a point in time which is relative to the cluster cycle reference. The offsets of all real-time applications in a real-time cluster are assigned in order to avoid a conflict of message transmission. All real-time applications start transmitting their respective time-triggered messages at their individual offsets. Each real-time application sends a time-triggered message once every individual period, after the offset time has elapsed. Figure 2.3 illustrates a cluster cycle of 6 milliseconds that contains the transmission instants of two real-time applications: appl1 and appl2. A period of 3 milliseconds and a 0 second offset are assigned to appl1, while appl2 has a period of 2 milliseconds and an offset of 0.5 milliseconds (in its cluster cycle).

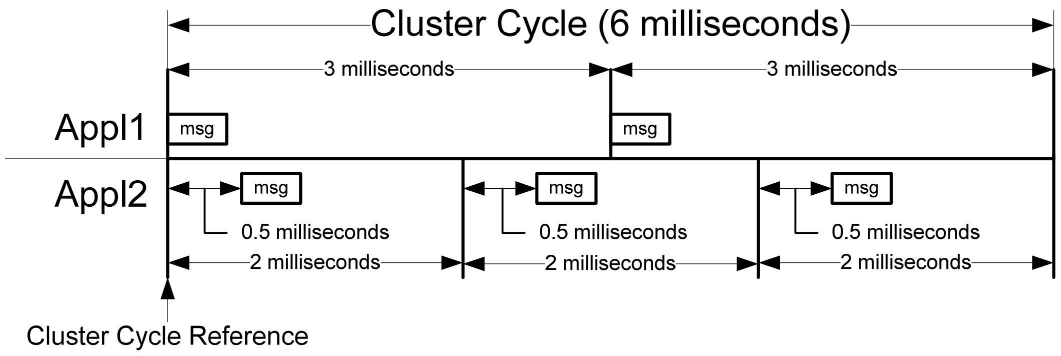


Fig. 2.3: A cluster cycle of two real-time applications (Appl1 and Appl2)

2.2.4 Message Permanence

A message that has arrived at a receiver node becomes *permanent* at a specific point in time, known as the *permanence point in time*. This point in time occurs when the receiver node knows that all messages sent to it, prior to the sending point in time of the current message, have arrived. Note that this is on the condition that no messages are lost during transmission. The permanence point in time can be considered to be equal to or greater than the sending point in time plus the maximum worst-case transport delay (of all communication channels in a real-time cluster). In TTEthernet, the concept of message permanence is used for a precise reestablishment of temporal order when receiving synchronization messages at a receiver node. Any receiver node in a TTEthernet network knows *a priori* the permanence point in time ($\text{Permanence}_{\text{PIT}}$) of a synchronization message. This can be calculated using Equation 2.2. The worst-case transport delay (TD_{WC}), which is equal to or greater than the maximum transport delay of synchronization messages from all communication channels (in the real-time cluster of a TTEthernet system), can be derived from either calculation or measurement. The point in time when a message is received at the receiver node is defined as the *receiving point in time* ($\text{Receiving}_{\text{PIT}}$). A synchronization message in a TTEthernet network contains the accumulated delay, called dynamic delay ($\text{D}_{\text{dynamic}}$), throughout the communication channel from the sender to the receiver.

$\text{Permanence}_{\text{PIT}} = \text{Receiving}_{\text{PIT}} + \text{TD}_{\text{WC}} - \text{D}_{\text{dynamic}}$ ………………………………….………...(2.2)

2.2.5 Determinism

In a safety-critical distributed system, the real-time communication system must support timely and deterministic communication in order to achieve fault-tolerance. For example, replica determinism [KG93] demands redundant nodes to take the same decision at approximately the same point in time [LM07]. With regard to time-triggered communication, Kopetz [KAGS05] defined *timely and deterministic* multicast transmission channels for safety-critical distributed systems with the following three properties:

1. *Given that a message is sent at the send instant $t_{send,}$ then receive instants $t_{receive}$ at all receivers of the (multicast) message will be in the interval ($t_{send} + d_{min}$, $t_{send}+d_{max}$), where d_{min} is called the minimum delay and d_{max} is called the maximum delay. The difference d_{max}-d_{min} is called the jitter of the transmission channel. d_{max} and d_{min} are a priori known characteristic parameters of the given transmission channel.*

2. *The receiver order of the messages is the same as the send order. The send order among all messages is established by the temporal order of the send instants of the messages as observed by an omniscient outside observer.*

3. *If the send instants of n (where n>1) messages are the same, then an order of the n messages will be established in an a priori known manner.*

For the first property, each message in a timely and deterministic communication channel must be transmitted with a small jitter that is known *a priori* in advance. For the second property, receivers must receive all messages in exactly the same order as the send order of senders, even in different communication channels. The last property ensures that messages transmitted at approximately the same instant and from different independent channels are received in the same order at all receivers in an *a priori* known manner. TTEthernet adopts the principle of time-triggered communication [KG93] on Ethernet to enable this determinism property.

2.3 Ethernet Technology

Ethernet is an open standard communication protocol. It was standardized by the Institute of Electrical and Electronics Engineers (IEEE) in the IEEE standard 802 part 3 [IEE05a]. Ethernet was invented in 1976 at the Xerox Palo Alto Research Center as a local area network, connecting: workstations, servers, and laser printers. The data transmission rate of this original Ethernet was 2.94 Mbps. This original Ethernet was developed from the concept based on the protocol *Aloha* [Spu00]. According to Abramson [Abr77], *pure Aloha* obtains a maximum channel utilization of approximately 18 percent, and channel utilization of *slotted Aloha* raises utilization up to 37 percent. These utilizations are low (less than 50 percent) because of the increasing rate of the collisions under the increasing load. The original Ethernet was an enhanced version of *Aloha*, i.e. the original Ethernet achieved better channel utilization than *Aloha*.

2.3.1 Ethernet Standard as IEEE 802.3

The original 10-Mbps Ethernet standard was first published in 1980 by the DEC Intel-Xerox vendor consortium [Spu00]. This Ethernet standard that was known as the *DIX Ethernet, and* was entitled: *The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications*. The *DIX Ethernet* contains the operation and specification of Ethernet for a single media system based on thick coaxial cable. The IEEE standard for open Ethernet was first published in 1985 with the title: *IEEE 802.3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*. The original IEEE 802.3 standard adopted the original DIX Ethernet standard with few modifications. The original IEEE 802.3 standard described media systems based on thick coaxial cable. The next development of the original IEEE 802.3 was to use a variety of transmission media (i.e. thin coaxial, twisted-pair, and fiber optic cables) in 10 Mbps systems. Higher speed versions of Ethernet were introduced and standardized as the supplement of IEEE 802.3 (i.e. in 1995 for 100 Mbps Ethernet -- Fast Ethernet, and in 1998 for 1000 Mbps -- Gigabit Ethernet).

2.3.2 Shared Ethernet

The original IEEE 802.3 standard describes 10 Mbps Ethernet in a shared communication channel. Ethernet that runs in a communication network based on a shared communication channel is called *shared Ethernet* [Spu00]. Shared Ethernet is where all host computers are connected to the same communication channel, and compete with one another for network access. The physical connection of a shared Ethernet network is illustrated in Fig. 2.4. Shared Ethernet uses a mechanism called *Carrier Sense Multiple Access/Collision Detection* (CSMA/CD) for gaining access to a shared communication channel. The Ethernet standard employs the *Manchester* encoding and decoding [Sei98] for data transmission in network communication mediums. The communication of a shared Ethernet network is *half-duplex communication*, denoting that a node can either send or receive a message at any one time.

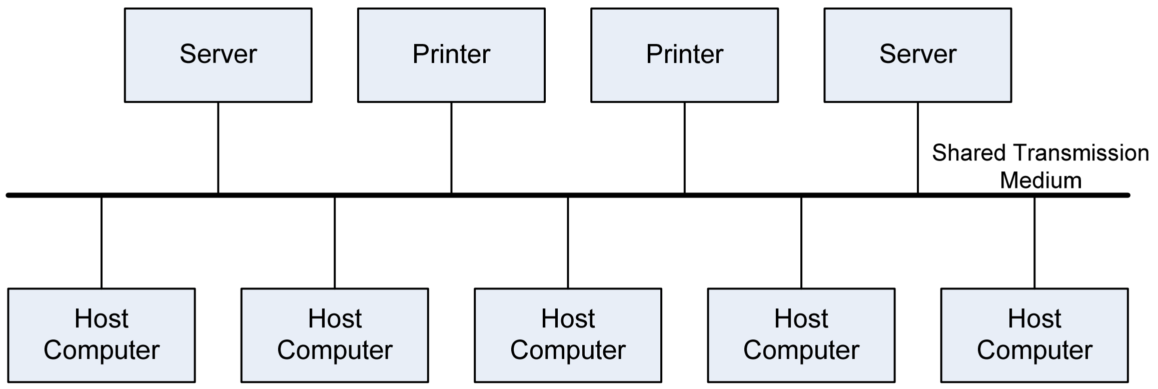
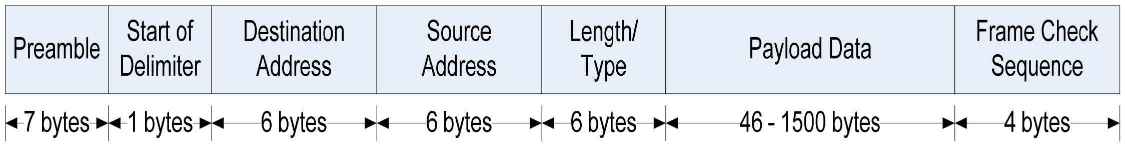


Fig. 2.4: A shared Ethernet network

CSMA/CD

The mechanism of CSMA/CD is briefly explained as follows. Once a node in a shared Ethernet network is ready to send an Ethernet frame, the node checks whether the shared medium is idle. If the shared medium is idle, the node will transmit the Ethernet frame immediately. Otherwise, the node has to wait until the shared medium becomes idle. A message collision may still occur however, if more than one node decides to transmit their respective messages at approximately the same instant. Each node has the *CSMA/CD* mechanism for detecting a collision occurrence in the shared communication medium. If a node that is sending a message detects a collision occurrence, the node will stop transmitting and wait for a random amount of time before retrying. The random amount of time is derived from the algorithm: *Truncated Binary Exponential Backoff* [IEE03]. In this algorithm, the IEEE 802.3 standard defines 16 as the maximum number of retransmission attempts.

Ethernet Frame Format



Preamble	Start of Delimiter	Destination Address	Source Address	Length/ Type	Payload Data	Frame Check Sequence
7 bytes	1 bytes	6 bytes	6 bytes	6 bytes	46 - 1500 bytes	4 bytes

Fig. 2.5: The Ethernet frame format

The IEEE 802.3 standard defines the fields in the Ethernet frame format as follows.

- *Preamble* Field: A 56-bit pattern of alternating ones and zeros. It is used for synchronizing the receiver’s clock to an incoming Ethernet frame.
- *Start of Delimiter (SFD)* Field: Indicates the beginning of an Ethernet frame. It consists of the following single byte: 10101011.
- *Destination Address* Field: Contains a 48-bit value which a receiving node uses to decide whether or not an incoming message has to be received or dropped. If a message is sent in unicast form, the D*estination Address* field of the message contains the Media Access Control (MAC) address of an intended destination node.
- *Source Address* Field: Contains the MAC address of a sending node.
- *Length/Type* Field: Indicates either the length of the payload data or the type of an Ethernet frame. If the value of the Length/Type field is less than or equal to 1500, this value denotes the number of the payload data. If the value is greater than 1536, this value is used to indicate which protocol is encapsulated in the payload data of an Ethernet frame. In the latter case this field is called an *EtherType* Field. The meaning of this field when it has a range between 1501 and 1535 is undefined. All values of *EtherType* can be accessed at the website [IEE]. Examples of *EtherType* values are: 0x8000 for *Internet Protocol version 4 (IPv4)*, and 0x891d for *Protocol Control Frame* (PCF, used in high-availability fault-tolerant TTEthernet).
- *Payload Data* Field: Contains either the payload data of an Ethernet frame, or encapsulates the sub-frame(s) of a higher-layer protocol(s), e.g. a frame of *IPv4*. The IEEE 802.3 standard defines the length of this field to be in a range between 46 and 1500 bytes. If the payload data contains less than 46 bytes, it will be padded with zeros until the minimum size of the payload data reaches 46 bytes.
- *Frame Check Sequence (FCS)* Field: A four-octet field used to verify that the information of a received frame was not corrupted during transmission. The method used for the verification is known as *Cyclic Redundancy Check (CRC)*. The field contains a 32-bit CRC value which is created by the MAC of a sending node, and then recalculated by the MAC of a receiving node. If the CRC value of a received frame does not match the recalculated CRC of the receiving node, the received frame will be identified as damaged, and thus be discarded. Otherwise, the received frame is valid, and will be forwarded to other processes in the receiver node.

  The IEEE 802.3 standard defines the minimum idle period between any two consecutive Ethernet frames as 96 bit times. The period between two consecutive Ethernet frames is called the *inter-frame gap (IFG)*, or *inter-packet gap (IPG)*. The 96 bit times of the IFG denotes the transmission time of 96 bits equal to: 9.6 microseconds for 10-Mbps

Ethernet, 960 nanoseconds for 100-Mbps Ethernet, and 96 nanoseconds for 1-Gbps Ethernet.

Repeaters and Hubs

A *repeater* is a two-port device that retransmits the electrical signal it receives as input, typically after it has been amplified. This is shown in Fig. 2.6(a). An *Ethernet hub*, more commonly referred to as simply a *hub,* is a multiport repeater, as depicted in Fig. 2.6(b). A network segment of coaxial-based shared Ethernet has a restricted size due to signal attenuation.

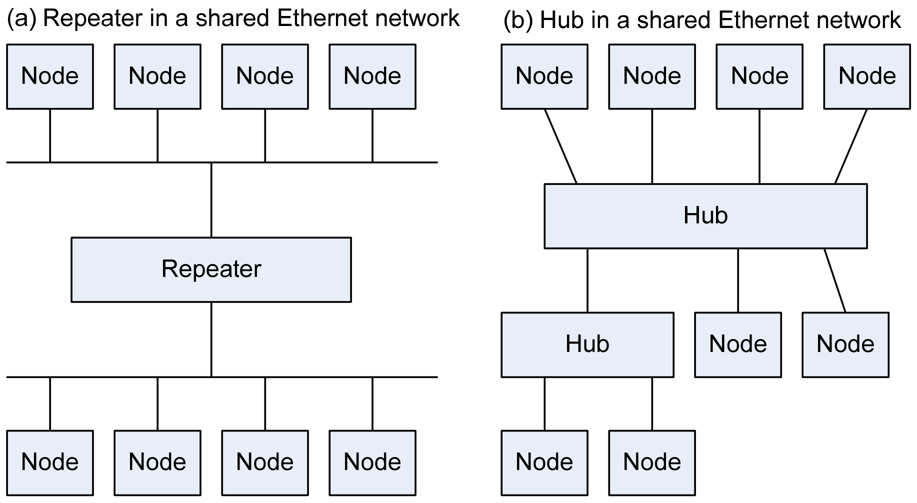


Fig. 2.6: Repeater (a) and hub (b) connection in a shared Ethernet network

To extend the network size, multiple network segments can be connected with repeaters or hubs. Such multiple network segments form a larger collision domain. In twisted pair-based shared Ethernet, a hub forms a network segment that connects end stations and network devices together, as depicted in Fig. 2.6(b).

In shared Ethernet, adding a network device (a hub or repeater) creates a larger collision domain. The larger the collision domain, the lower bandwidth utilization will be. This is because there are more devices that can try to send an Ethernet frame at any one time.

2.3.3 Switched Ethernet

The collision problem in a shared Ethernet network is resolved by means of *switched Ethernet* [Fei00] [Puz02]. In switched Ethernet, an Ethernet switch is used instead of a hub to restrict the collision domain into only one node, and one connected port of an Ethernet switch. This is shown in Fig. 2.7. No collisions occur in switched Ethernet networks. Switched Ethernet allows all devices full bandwidth utilization, i.e. a node can send out an Ethernet message without being aware of any collision with other Ethernet frames. Note that this is regardless of Ethernet overhead or inter-frame gap. This feature of switched Ethernet implies that the CSMD/CD mechanism in shared Ethernet is no longer required.

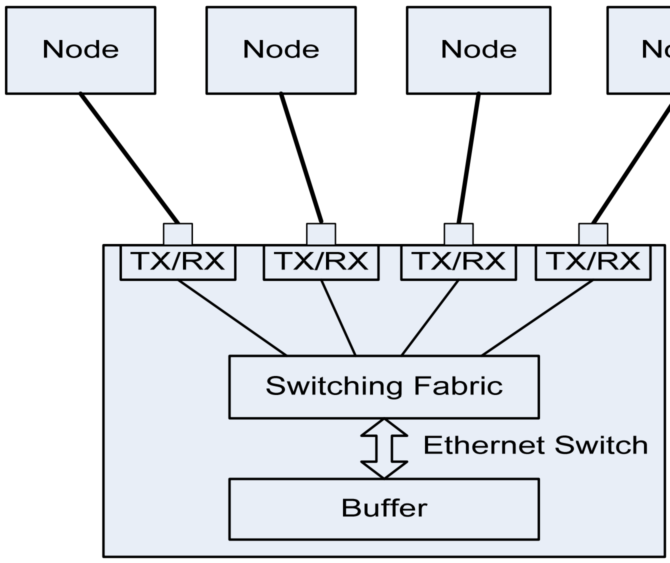


Fig. 2.7: An Ethernet switch with 4 ports

In contrast to *half-duplex communication* in shared Ethernet, switched Ethernet allows nodes to have their dedicated bandwidth on point-to-point connections support *full-duplex communication*. This means that a node in a switched Ethernet network can send and receive Ethernet frames simultaneously, without any collision occurrence. The typical switching method in a COTS Ethernet switch is the *Store and Forward* method [LL02], and is explained as follows: Once an Ethernet switch has received an Ethernet message from a sender node, it verifies the CRC value in the FCS field of the Ethernet frame with the calculated one. If both CRC values are the same, the Ethernet switch checks the address table regarding which port(s) the Ethernet frame has to be relayed to. The Ethernet frame will be discarded if its CRC is not identical to the calculated one. After the Ethernet switch knows the output port(s), it stores the Ethernet frame to the output port's buffer(s). The Ethernet controller(s) of the output port(s) forwards the Ethernet frames from the buffer(s) on a *First Come First Served* basis. The significant drawback of using an Ethernet switch is that its buffer could overflow in cases of high load conditions (i.e. some of the Ethernet frames receiving at the Ethernet switch could be dropped).

2.4 Ethernet Performance Metrics

Switched Ethernet no longer requires the *CSMA/CD* mechanism of shared Ethernet. Switched Ethernet obtains full channel utilization of a link regardless of the overhead of an Ethernet frame, or inter-frame gap (IFG). There are many performance aspects of switched Ethernet, i.e. throughput, latency, frame loss rate, back-to-back frames, jitter, and bit error rate [Flu08]. Most studies presenting results on the performance of a switched Ethernet network are concerned with bandwidth, i.e. *throughput* [PJ05] [PA02] [ASV00] [FO00]. Besides throughput, another significant aspect of performance of switched Ethernet involved with timing performance is *latency* [LH04] [LDK10]. Latency is referred to via several terms, e.g. end-to-end delay [HF04] [LL02], round-trip time [VVT06], or message delay [KSS07]. Some applications in switched Ethernet networks (e.g. voice over IP, and online gaming) demand timing guarantees in message transmission. The test report [EAN09] of the European Advanced Networking Test Center (EANTC) shows the performance of a gigabit Ethernet switch (Arista's 7148SX 48-port, 10GbE switch) by two performance characteristics: throughput and latency. In this thesis, throughput and latency are the key performance measures of standard Ethernet in a TTEthernet system. Therefore, the basic concept of throughput and latency in a switched Ethernet network are presented in detail as follows:

2.4.1 Throughput

Network throughput is a measure of how much data can be sent across a network per unit of time. Throughput also denotes the average rate of successful message delivery over a communication channel. The throughput of a given protocol layer is defined as the *number of bits* per second (bps) transferred from a given layer to its upper layer, as a result of communication. In case of the data link layer (MAC layer of Ethernet) of the OSI reference model, throughput considers only data which are forwarded to the OSI layer above (the network layer). This is always less than the bandwidth utilization at the physical layer due to the overhead caused by protocol headers (added at the MAC layer). The detail of channel utilization is explained in the *Channel Utilization* section. It is worth noting that throughput and bandwidth are different things. Bandwidth (also called channel capacity) is a measure of the data-carrying capacity of a data communication channel, and is thus a constant value. Throughput values however tend to vary. The maximum possible throughput of switched Ethernet is always less than the bandwidth. This is because throughput does not count the overheads of Ethernet frames (i.e. Ethernet headers, Inter-frame gaps). Throughput can be difficult to approximate as it can be negatively affected by interference and by other traffic.

Maximum Theoretical Throughput

With regard to switched Ethernet, the maximum theoretical throughput can be calculated with respect to the payload data size of an Ethernet frame (see Equation 2.3).

Given that:

The maximum theoretical throughput = MTT

Payload Data size in bits = PD

Preamble size in bits = Pr = 56 bits

Start of Delimiter in bits = SFD = 8 bits

Ethernet Header size in bits = EH = 112 bits

Frame Check Sequence size in bits = FCS = 32 bits

Inter-frame Gap size in bytes = IFG = 96 bits

Bandwidth = Ethernet bandwidth

$$MTT = PD/(Pr+SFD+EH+PD+FCS+IFG)*Bandwidth \qquad (2.3)$$

Note that the maximum size of the payload data of an Ethernet frame is 1,500 bytes, and the minimum size is 46 bytes. Considering the bandwidth of Fast Ethernet (100 Mbps), the maximum theoretical throughput with regard to the maximum size of an Ethernet frame can be calculated using Equation 2.1 as 97.53 Mbps. In case of the minimum size of an Ethernet frame, the maximum theoretical throughput is 54.76 Mbps.

Channel Utilization

Channel utilization is also called bandwidth utilization. It is the total number of bits transferred at the physical layer to communicate a certain amount of data to a higher layer (the data link later for Ethernet), divided by the time taken to communicate the data. Channel utilization is expressed as a percentage of the transmission data rate at the physical layer. For the channel utilization of switched Ethernet, the data in bits is all bits of all fields in an Ethernet frame transmitted at the physical layer (i.e. Preamble, Start of Delimiter, Ethernet header, Payload data, and Frame Check Sequence). Channel utilization counts all transmitted frames irrespective of whether they are corrupted or correctly received. Note that there are two main points where throughput differs from channel utilization: throughput takes only data in the payload field of an Ethernet frame into account; and counts only frames that are received at the intended destinations correctly. Channel utilization is typically expressed as a percentage of the data transmission rate with respect to channel capacity (Ethernet bandwidth). In switched Ethernet, the maximum channel utilization is the maximum possible data rate at the physical layer, and can be calculated with respect to Ethernet frame size (see Equation 2.4).

Given that:

The maximum channel utilization = MCU

Payload Data size in bits = PD

Preamble size in bits = Pr = 56 bits

Start of Delimiter in bits = SFD = 8 bits

Ethernet Header size in bits = EH = 112 bits

Frame Check Sequence size in bits = FCS = 32 bits

Inter-frame Gap size in bytes = IFG = 96 bits

MCU = (Pr+SFD+EH+PD+FCS)/(Pr+SFD+EH+PD+FCS+IFG)*100%..................(2.4)

Note that the maximum and minimum size of the payload data in an Ethernet frame is 1,500 bytes and 46 bytes, respectively. The channel utilizations of maximum-sized Ethernet frames and minimum-sized Ethernet frames are 99.22% and 85.71%, respectively. It is notable that channel utilization of maximum-sized Ethernet frames is superior.

2.4.2 Latency

Latency in a communication network refers to the timing of data transfer, i.e. it is defined as the time it takes for a message to traverse a network from transmitter (source) to receiver (destination). Latency is an important aspect of performance, and is demanded in delay-sensitive applications, e.g. voice over IP, video conferencing [SJR05] [BGM10], or online gaming [LDK10]. In some cases, excessive network latency can cause such delay-sensitive applications to be unusable. Network latency is referred to using various terms, such as: end-to-end latency (or end-to-end delay) [HF04] [LL02] [SJR05], round-trip latency (or round trip time) [VVT06], or message delay [KSS07]. These terms refer to the timing performance of message delivery on a network, even if they do not all share a common definition. For example, end-to-end latency is the time it takes for a message to get from source to destination, whereas round-trip latency is the time it takes for a message to get from source to destination and then back to source. In a switched Ethernet network, the transport delay of an Ethernet frame transmitting from the Ethernet controller of a source node to the Ethernet controller of the intended destination node, is affected by the following sources of latency: 1) Transmission latency at a transmitter, 2) Wire-line latency, 3) Switch fabric latency, 4) Frame queuing latency in Ethernet switches, and 5) forwarding latency at the Ethernet switches. These latency sources are depicted in Figure 2.8.

(1) *Transmission Latency at a transmitter (L_T)*: The transmission latency (L_T) (also called the transmission time) is the time it takes for an entire Ethernet frame to transmit from the physical layer of the Ethernet controller at a transmitter (at the source node in Figure 2.8) into the connected transmission line. It can be calculated using Equation 2.5.

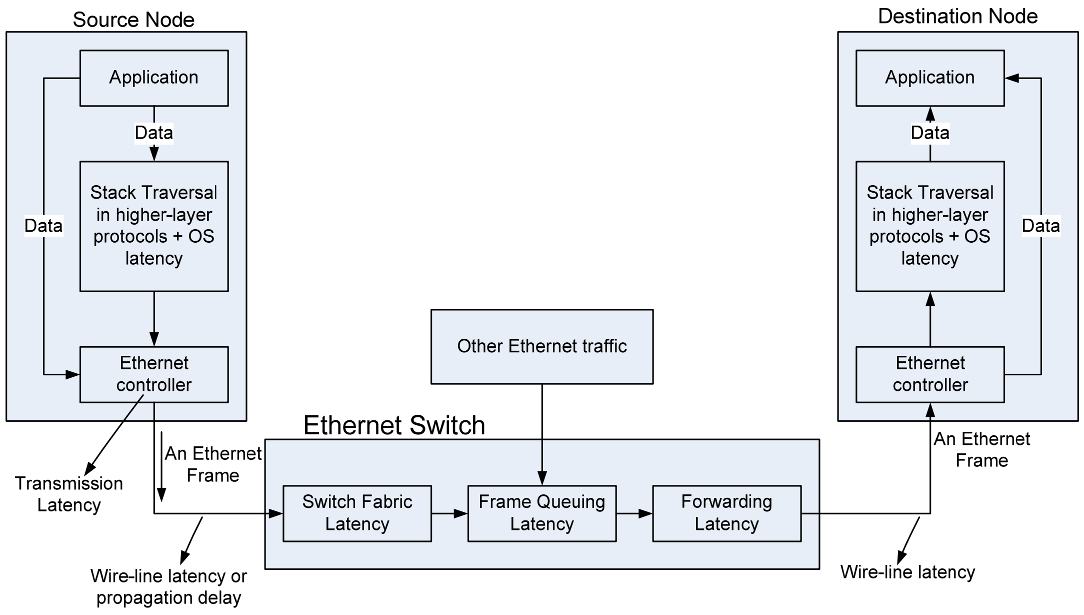


Fig. 2.8: Sources of latency in a switched Ethernet network

$$L_T = FS/TR \quad \text{……………………………………}(2.5)$$

, where L_T is the transmission latency, FS is the frame size in bits, and TR is the transmission rate in bits per second (or Ethernet bandwidth).

(2) *Wire-line latency (L_{WR})*: The wire-line latency is the *propagation delay* of a transmission line. It denotes the time it takes for a bit of an Ethernet frame to travel over a transmission line. Using copper cable as a transmission line, a signal can travel with a transmission speed of

approximately 2/3 of the speed of light ($3x10^8$ m/s) in vacuum [Kop97]. The wire-line latency can be calculated using Equation 2.6.

$$L_{WR} = TL/TS \qquad (2.6)$$

, where L_{WR} is the wire-line latency in seconds, TL is the length of a transmission line in meters, and TS is the transmission speed of a signal travelling over a transmission line. For example, if the length of a copper cable is 100 meters, the wire-line latency is approximately 0.5 microseconds ($L_{WR} = 100 / (2/3x3x10^8)$). Note that in local area networks, this form of latency is negligible compared with the other contributors to latency [Rug08].

(3) *Switch Fabric Latency (L_{SF})*: The switch fabric is the component of an Ethernet switch that moves an incoming Ethernet message from an input port to an output port(s). Typically the switch fabric of an Ethernet switch contains the following functions: the store and forward engine, the MAC address table, and the VLAN (Virtual LAN) functions [Rug08]. Therefore, the switch fabric of an Ethernet switch introduces delays for message transport in a switched Ethernet network. The switch fabric latency value of an Ethernet switch must be specified in its product datasheet, e.g. the switch fabric latency on RuggedSwitch® products is 5.2 microseconds [Rug08]. Switch fabric latency may be indicated by the forwarding rate, e.g. the forwarding rate of the Cisco Catalyst 3560G-48TS Switch based on 64-byte packets is 38.7 Mpps (million packets per seconds) [Cis09].

(4) Frame Queuing Latency *(L_{FQ})*: Ethernet switches use queues to eliminate the problem of frame collisions that exist in shared Ethernet networks. The queue of an Ethernet switch performs incoming Ethernet frames on a *First Come First Served* basis. Frame queuing introduces a non-deterministic factor to latency because it is very difficult to predict exact Ethernet traffic patterns on a network. Even though message prioritization was introduced in IEEE 802.1Q [IEE05b] to increase the quality of service in an Ethernet switch, the unpredictable queuing of multiple frames with the highest priority level is still an issue (in non-deterministic latency). Additional problems include that if a low-prioritized Ethernet frame has already started transmission, that transmission must be completed before the Ethernet switch begins transmitting any new, high-prioritized Ethernet frame(s). Despite the difficulty in predicting frame queuing latency, it can be calculated (using Equation 2.7) if the number of preceding Ethernet frames (N_Q) existing in the queue of an Ethernet switch is known. Note that if there are no Ethernet frames buffering the queue of an Ethernet switch, then the frame queuing latency (L_{FQ}) is equal to zero.

$$L_{FQ} = \sum_{i=1}^{N_Q} (L_F)_i \qquad (2.7)$$

, where $(L_F)_i$ is the Forwarding Latency of the i^th Ethernet frame in the queue, and N_Q is the number of Ethernet frames in the queues.

(5) *Forwarding Latency (L_F)*: Forwarding Latency is the *transmission latency* of an output port of an Ethernet switch. The calculation is the same as for the transmission latency of a transmitter.

$$L_F = FS/TR \qquad (2.8)$$

The **total network latency (L_{TOTAL})** of a switched Ethernet network, starting from the time when an Ethernet frame is transmitting from the source node, up until the time when the entire Ethernet frame has arrived at the destination node, can be calculated using Equation 2.9:

$$L_{TOTAL} = L_{T(SOURCE)} + L_{WR(total)} + \sum_{i=1}^{N_{SW}} \left(L_{SF} + L_{FQ} + L_F\right)_i \qquad (2.9)$$

, where the number of Ethernet frames in each of the queues of the Ethernet switches is known.

2.4 Mixed-criticality systems

TTEthernet supports *mixed-criticality systems*, i.e. applications with different criticality levels interacting and coexisting in a single communication infrastructure. The time-triggered communication paradigm [KG93] enables deterministic communication in TTEthernet. This means that TTEthernet can obtain temporal guarantees for hard real-time applications, and support fault-tolerant real-time systems [KAGS05]. TTEthernet provides a set of time-triggered services that are implemented on top of standard Ethernet (IEEE802.3) [IEE05a]. In addition to time-triggered communication, TTEthernet was also designed to be fully compatible with standard Ethernet. This means that commercial-off-the-shelf (COTS) Ethernet devices can, without modification, directly connect to TTE switches for transmission of standard Ethernet traffic. A COTS Ethernet device, which transmits standard Ethernet frames in an event-triggered manner [HRB07], cannot interfere with the timely behavior of time-triggered messages. TTEthernet allows applications with bandwidth constraints (e.g. multimedia applications, AFDX applications [Con05]) to coexist in the same communication infrastructure. Thus, TTEthernet classifies traffic for mixed-criticality applications into three different traffic classes: time-triggered traffic, rate-constraint traffic, and best-effort traffic. The details of these traffic classes are given in the TTEthernet section in Chapter 3 – *State of the art*.

2.5 Discrete Event Simulation

There are two main types of simulation: continuous simulation and discrete event simulation [OB09]. Continuous simulation is suitable for systems where the state variables change continuously with respect to the progression of time. Discrete event simulation is suitable for systems where the state variables change in discrete times (discrete time steps). In a discrete event simulation, the simulation time advances as the next event occurs, i.e. the duration of each time step is uncertain because the next event could occur at any point in simulation time. In a continuous simulation run using a digital computer, the size of the time steps are equal, and so sufficiently small that there are no transitions within the system between any two consecutive time steps. In this thesis, we found discrete event simulation to be the most suitable for a TTEthernet system. This is because the continuous simulation issue regarding time step transitions is not a problem in a discrete event simulation. In addition to this advantage, a discrete event simulation can also increase the efficiency of executing a simulation model for a TTEthernet system (in which event-triggered traffic may occur at any point). Therefore, we have developed a simulation model of a TTEthernet system based on discrete event simulation.

Java-based discrete event simulation

We use the Java runtime environment and the Java library *J-Sim* [JSim] for implementing our TTEthernet simulation model. *J-Sim* is an object-oriented library for the development of java-based discrete event simulations. It is written purely in Java, is freeware, and is based on known principles from the Simula language [Kac02].

With regard to discrete event simulation, every simulation has its own time, called the *simulation time*. The simulation time represents a reference clock time, i.e. it has a drift rate of zero. One simulation step in a distributed event simulation executes one event in the common event list. This event occurs at one exact point in the simulation time. Two or more simulation steps can be executed at the same simulation time. New simulation steps can be created dynamically. Once the execution of the current simulation step has finished, the next one to be executed is determined from the common event list. In J-Sim, all simulation events are stored in the common event list, called the *simulation calendar*. An event is an object containing information about an executed point in the simulation time (timestamp) and a process that will be run at that time. When a new event is put into the simulation calendar, it is sorted in ascending order of the simulation time. Therefore, the event with the lowest timestamp will be removed and executed first. Each event is put in the simulation calendar by either itself or other events with a known point in the simulation time. J-Sim provides a *step()* method in the class *JSimSimulation*, for the execution of a single simulation step. The event processes in the simulation calendar are instantiated from the classes inherited from the class *JSimProcess*. Each event process has a *life()*

method, containing the code describing its process behavior. There are five main methods the processes use to interact with the simulation calendar. These are described as follows:

- *Activate (point in simulation time):* Inserts a new event into the simulation calendar. The process which is currently executing can use this method to activate any other processes.
- *Hold (time interval): Temporarily* suspends the currently running process (in the simulation calendar) for the specified time interval. This method can only be invoked by the currently running process.
- *Passivate ():* Suspends the currently running process by dropping it from the simulation calendar. The process stays out of the simulation calendar until another process invokes its *activate* method. This method can only be invoked by the currently running process.
- *Cancel ():* Removes a *specified* process from the simulation calendar. The currently running process can invoke this method on itself as well as on any other processes. A process that was invoked by this method stays out of the simulation calendar until another process invokes its *activate* method.
- *Reactivate ():* Re-inserts the currently running process into the simulation calendar. This occurs at the current point in simulation time. All other events at the same point in simulation time will be executed beforehand.

CHAPTER 3

State-of-the-art

Ethernet, which is widely used in home and office computer networks, was originally not designed for real-time communication. Standard Ethernet is not suitable for real-time communication, as it is not a timing-deterministic communication protocol [Dec05] [Fel05]. With the advancement of modern Ethernet technology, bandwidths are extremely high (as seen in Fast and Gigabit Ethernet technologies). This is especially true when compared with other modern communication protocols, such as: Flexray from the automotive domain (10 Mbps) [NHB05], Profibus from the industrial automation domain (12 Mbps), and ARINC 429 from the aerospace domain (100 kbps) [SV08]. Besides high Ethernet bandwidth, the cost of commercial-off-the-shelf (COTS) Ethernet devices is low. Therefore Ethernet, which is conventionally used in *open-world systems* [KAGS05], is attractive for use in real-time communication networks demanded by *closed-world systems* [KAGS05] (e.g. fly-by-wire systems, fieldbus systems, and so on).

Over the past decade many solutions have been developed to adopt standard Ethernet to be capable of providing the temporal guarantees required by real-time communication networks [Fel05] [Dec05]. Such solutions (i.e. standard Ethernet-based protocols augmenting with temporal guarantees) are called r*eal-time Ethernet* [Fel05] [WK06]. Most solutions of real-time Ethernet have emerged from the industrial automation domain. Examples of such solutions are: PROINET IO, EtherNet/IP, Ethernet POWERLINK, EtherCAT, Profinet and SERCOS III [Dec05] [Fel05]. In the avionics domain, standard Ethernet was also used in safety-critical real-time systems. Two such systems are AFDX [Con05] and TTEthernet [TTE08]. We have selected interesting protocols of *real-time Ethernet* that are dominant in the current market and academic research. These are: EtherNet/IP, PROFINET, MODBUS-TCP, Ethernet POWERLINK, EtherCAT, SERCOS III, AFDX, and TTEthernet. The following significant aspects of these protocols will be described in the sections that follow: protocol structure, functionality, performance, and their ability to coexist with legacy Ethernet devices.

3.1 EtherNet/IP

EtherNet/IP (Ethernet/Industrial Protocol) [Pau01][Odv06] is an industrial communication protocol developed and maintained by ControlNet International (CI), Open DeviceNet Vendors Association (ODVA), and the Industrial Ethernet Association (IEA). EtherNet/IP is an open application-layer protocol that runs on top of the transport layer (TCP or UDP) of the TCP/IP protocol model. It uses the open object-oriented CIP (Common Industrial Protocol) as the application layer, as depicted in Fig. 3.1. The CIP is useful for interoperability between an industrial communication network (an Ethernet/IP, ControlNet or DeviceNet network) and a standard-Ethernet network. EtherNet/IP offers a real-time solution by using strict recommendations [Ana02], but it is still not deterministic. It is also not a hard real-time communication protocol due to both: variable delays in the Ethernet/IP protocol stack, and uncertain queuing delays occurring at the Ethernet switches.

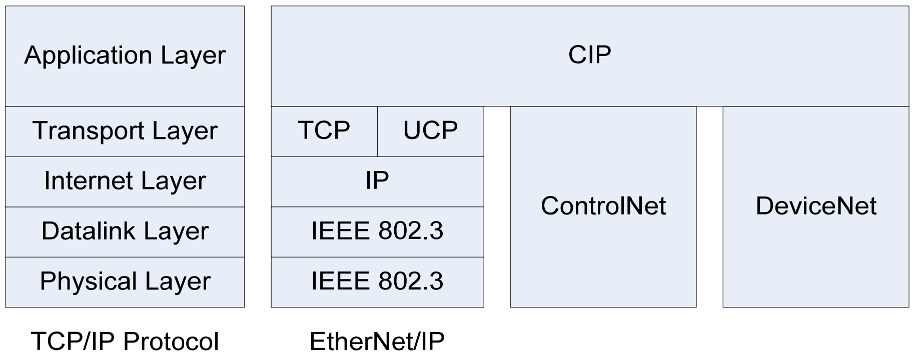


Fig. 3.1: The EtherNet/IP protocol stack and TCP/IP protocol model

3.1.1 Structure and Functionality of EtherNet/IP

EtherNet/IP is an application-layer protocol that runs on top of the transport layer of the TCP/IP protocol model. It employs both TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) for message transmission. At the operation initialization of Ethernet/IP, TCP is used for establishing a connection between distributed devices. An EtherNet/IP message based on TCP is called an *explicit message*, as shown in Fig. 3.2. The transmission of an explicit message is reliable, but there is no timing guarantee. This is due to the potential retransmission of an explicit message in the TCP/IP mechanism. EtherNet/IP uses UDP for real-time communication in both unicast and multicast transmission forms. An Ethernet/IP message based on UDP is called an *implicit message*, as shown in Fig. 3.2. Implicit messages are exchanged based on the producer-consumer model, where a single transmitting device (producer) sends a message to many receiving devices (consumers) at a time. This producer-consumer model performs its operation based on an Internet Protocol (IP) multicast service (i.e. the service of the producer-consumer model is mapped to an Ethernet multicast service). An implicit message contains only time-critical data, whereas an explicit message includes command information. According to [Pau01], there are four kinds of implicit messages available in the CIP specification: polled, change of state, cyclic and strobed. Cyclic messages are preferred for implicit message exchange in EtherNet/IP networks. They are produced by a producer device according to a predetermined schedule. A consumer device accepts an arriving cyclic message if and only if the connection between those devices has been established. The mechanism of Ethernet/IP connection establishment provides a timeout mechanism that can detect a data delivery problem before any real-time messages are transmitted. Although Ethernet/IP employs implicit messages for real-time communication, the transport delays of implicit messages in an Ethernet/IP network are uncertain. This is due to variable delays in the Ethernet/IP protocol stack, and queuing delays occurring at Ethernet switches. With the advent of extremely high Ethernet-bandwidth (i.e. Fast and Gigabit Ethernet), EtherNet/IP networks are suitable for some real-time applications, but not for high-speed real-time systems (e.g. high-speed motion control systems).

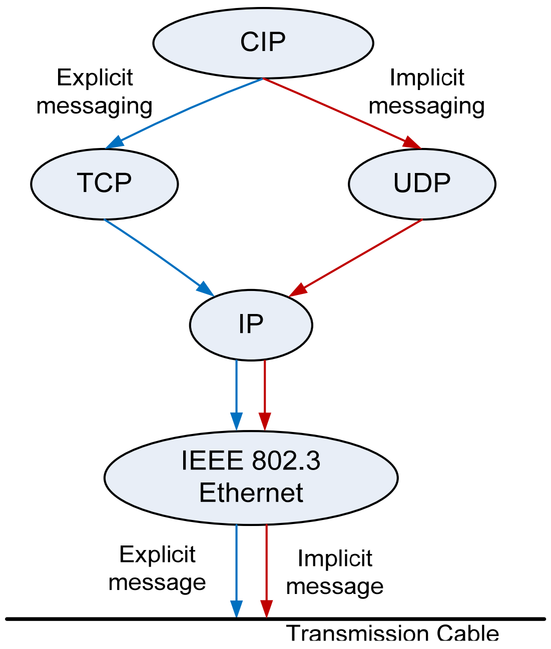


Fig. 3.2: Explicit and implicit messaging of Ethernet/IP

CIP (Common Industrial Protocol)

Ethernet/IP extends commercial-off-the-shelf standard Ethernet with CIP, as shown in Fig. 3.2. CIP proposes an interoperability platform for manufacturing applications from different venders (i.e. so that devices produced by different vendors can communicate with each other without modification). Therefore, messages from devices in EtherNET/IP networks can transmit to devices in other kinds of networks (e.g. DeviceNet and ControlNet networks) without using a gateway device. CIP makes devices from different vendors compatible with each other by using an object model approach. The object model used in CIP is called the *CIP object model*. CIP also provides a suite of services for: control, configuration, and data collection. These services can be transported via either implicit or explicit messages. Each device from a manufacturer is defined using their identification information, called the *device profile*.

CIP Object Model

CIP is an object-oriented protocol at the application layer of the TCP/IP protocol model. Each CIP object contains attributes (data), services (commands), connections, and behaviors (reactions to events). Objects that behave identically in two or more devices are grouped together. The grouping of these objects in a device is referred to as the object model. The concept of the object model supports interoperability of type-identical devices from different vendors. In addition, the CIP object model (based on a producer-consumer model) provides more efficient resource utilization than a source-destination based model. This is because a data message can be simultaneously transmitted from a sender device (the producer) to many receiver devices (the consumers).

Device Profile

In CIP networks, which consist of devices from many vendors, CIP supports interoperability between devices by defining standard groupings of objects as *device profiles*. Devices that have the same profile behave identically, even if they are supplied by different vendors.

3.1.2 Performance of EtherNet/IP

EtherNet/IP adopts advanced Ethernet technology (Fast and Gigabit Ethernet) for industrial networks without modification. Therefore, the mechanism of a standard Ethernet switch determines the real-time communication behavior of EtherNet/IP. Ethernet switches in EtherNet/IP networks are responsible for relaying all incoming Ethernet frames in multicast form (i.e. incoming Ethernet frames are forwarded to all switch ports except the switch port receiving the frames node). Forwarding an incoming Ethernet frame in multicast form degrades network performance (i.e. forwarding Ethernet frames are not needed in some Ethernet switch ports, and they could interfere with other traffic). EtherNet/IP takes advantage of some advanced features of modern Ethernet technology to reduce undesired traffic loads from multicast transmission. The following recommendations [Ana02] regarding EtherNet/IP were made in order to enhance its real-time capability.

➢ *VLAN (Virtual Local Area Network)*: Sub-networks in an Ethernet switch can be separated using VLAN features, as depicted in Fig. 3.3. Therefore, the real-time messages transmitted in multicast form are confined in predefined individual sub-networks in the Ethernet switch. This guarantees that real-time messages in one sub-network cannot interfere with other sub-networks.

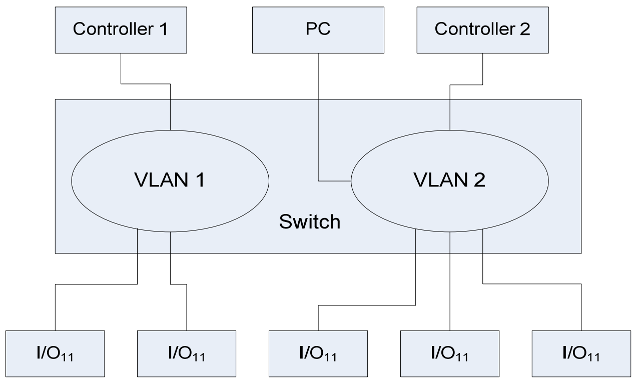


Fig. 3.3: An Ethernet switch configured with two VLANs

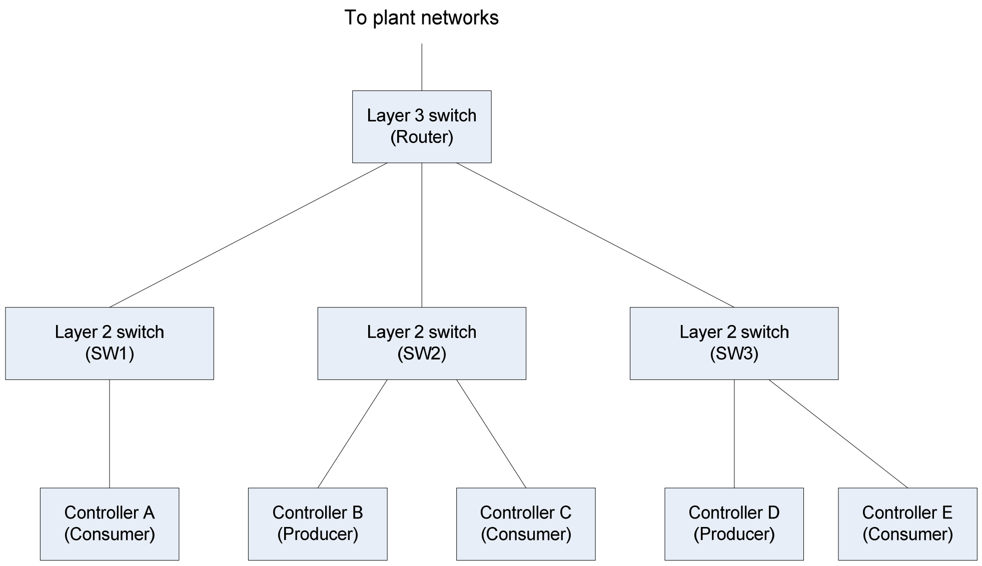


Fig. 3.4: An EtherNet/IP network with producer-consumer communication

- *IGMP (Internet Group Management Protocol) snooping*: The IGMP snooping in Layer 2 switches eliminates undesired IP multicast traffic generated inside the VLAN. An example is shown in Fig. 3.4. This figure depicts IP multicast messages from Controller B destined for Controller C, and IGMP snooping being activated in the *SW2* switch. The IP multicast messages are confined to the switch, to which both controllers are connected.

- *TTL (Time-To-Live):* The TTL thresholds of IP messages can be set in Layer 3 switches. An Example is shown in Fig. 3.4. In this figure, IP multicast messages forwarded by the Layer 3 switch to the Layer 2 switches are checked against their TTL threshold. The IP multicast messages will be discarded if their TTL is less than a TTL threshold specified in the Layer 3 switch. On the contrary, IP multicast messages in which their TTL values are 1, and from the Layer 2 switches as shown Fig. 3.4 will be blocked at the Layer 3 switch.

With these recommendations, EtherNet/IP devices can achieve real-time performance without modification to existing COTS Ethernet hardware. According to [Ana02], the currently achievable end-to-end response time in an EtherNet/IP network consisting of eight producers and one consumer is 7 ms (with use of the aforementioned recommendations). However, the recommendations only reduce undesired traffic in an EtherNet/IP network. EtherNet/IP networks are still unable to handle a high amount of real-time traffic in a deterministic manner. As a result, EtherNet/IP networks can be used for real-time applications if and only if the amount of real-time traffic is limited.

3.2 PROFINET

PROFINET [PI09] is an industrial Ethernet standard which was devised by PROFIBUS International (PI). PROFINET aims to adapt Ethernet for all types of industrial automation networks (information-level networks, control-level networks and field-level networks). Since there are various real-time requirements for industrial automation applications, PROFINET provides two high layer protocols, namely *PROFINET CBA* (Component Based Automation) and *PROFINET IO*. Both PROFINET CBA and PROFINET IO have different real-time achievements, and are therefore employed at different levels of factory communication systems. PROFINET CBA is suited for component-based machine-to-machine communication via TCP/IP, whereas PROFINET IO is used for fast data exchange between controllers and field devices (especially in time-critical synchronous applications such as motion controllers). With reference to [PAS06], PROFINET defines three types of traffic: non real-time traffic, real-time class 1, and real-time class. These are shown in Table 3.1. PROFINET CBA handles both non real-time traffic and traffic of real-time class 1, whereas PROFINET IO handles traffic of both real-time class 1 and class 2. Note that performance of real-time class 2 is only possible with PROFINET IO. The real-time class 2 is known as *isochronous real-time,* and is based on a highly precise and synchronized cycle of message transmission.

Table 3.1: Real-time classes of PROFINET [PAS06]

Type of traffic	Periodicity/Reaction times	Jitter (%)
Non real-time	≥ 100 ms	≥ 100
Real-time class 1	≥ 5 ms	≥ 15
Real-time class 2	≥ 250 µs	≤ 0.4 (1µs)

Note that Jitter is computed with respect to periodicity.

3.2.1 Structure and Functionality of PROFINET

There are two sub-protocols of PROFINET: PROFINET CBA and PROFINET IO. Both of these are described in the sections that follow.

3.2.1.1 PROFINET CBA: PROFINET CBA is a high-level network protocol for industrial applications. It adopts Ethernet hardware and software for industrial networks without modification. PROFINET CBA focuses on *technological components* (i.e. all components act independently and coordinate their tasks to form an integrated system). A PROFINET CBA application is a device-specific implementation of the PROFINET CBA Runtime Object Model. The Runtime Object Model describes the objects within a device (their interface and methods). The basic components of a Runtime Object Model (depicted in Fig 3.5) are: Physical Device (PD), Logical Device (LD), Runtime Auto (RT-Auto), and Active Control Connection Object (ACCO). The component hardware that connects to an Ethernet network is represented by a PD. An application (e.g. a sensor or actuator task) in a device is represented by a LD. In a single PD, there can be many LDs. A Runtime Auto represents an automation function. Since a LD can have many functions, any LD could have several RT-Autos. An active control connection object (ACCO) provides configurable connections to every RT-Auto object.

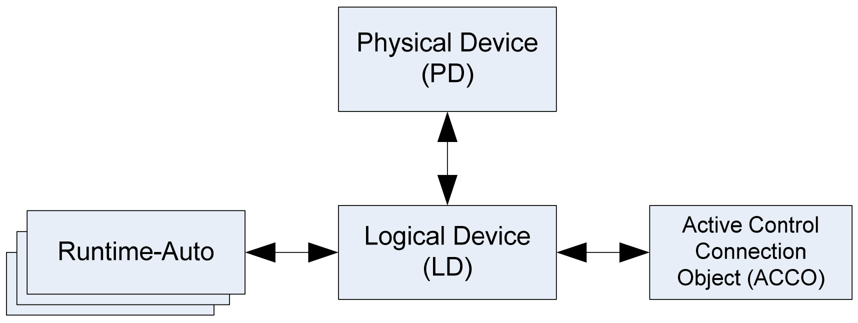


Fig. 3.5: Runtime Object Model of a PROFINET CBA device

Besides the Runtime Object Model, PROFINET CBA also supports DCOM (Distributed Component Object Model) [BOX98]. The DCOM extends COM (Component Object Model) by including all functionalities for distributing data across a network in a reliable, secure, and efficient manner. PROFINET CBA interfaces to the outside world using a COM interface. A COM interface is a collection of Properties, Methods, and Events. Properties are the public data of a PROFINET CBA object. Methods define services and functions provided by a COM interface. Events signal a change in state for the interface. PROFINET CBA supports both non real-time and real-time class 1 communication. The communication behavior of PROFINET CBA (based on the DCOM model) is explained as follows: The DCOM model exchanges data according to a producer-consumer model. The data exchange occurs only when a value at the producer side has changed. A message transmission of PROFINET CBA begins with a cyclic event, called the QoS (Quality of Service) event. As shown in Fig. 3.6, when a QoS event occurs (1), the ACCO producer invokes the RT-Auto for checking any change of the values (2). If there is a change, the new data is transferred to the ACCO consumer (3) over the DCOM, TCP/IP stack and Ethernet networks. Subsequently, the ACCO Consumer receives the data from the DCOM at the consumer side (4), and in turn deliveries them to the RT-Auto (5 - the application function).

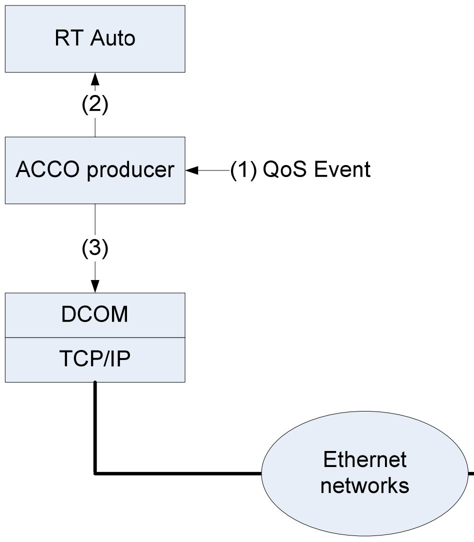


Fig. 3.6: Data exchange in PROFINET CBA over DCOM

3.2.1.2 PROFINET IO: PROFINET IO was introduced to satisfy stringent timing requirements in PROFINET networks. It can be used to connect distributed I/O devices for fast data exchange, e.g. for motion control applications. PROFINET I/O is designed to support all types of traffic: non real-time (Non-RT), real-time class 1 (RT Class 1), and real-time class 2 (RT Class 2). PROFINET IO classifies devices into the following three types:

1. IO-Controller: An intelligent device such as a programmable logic controller. An IO-Controller performs automation tasks.
2. IO-Device: A distributed I/O field device such as a sensor, an actuator, or an electronic terminator.
3. IO-Supervisor: A device that exchange both configuration and diagnosis data with IO-controllers and IO-devices. An IO-Supervisor can be a personal computer or a human machine interface (HMI) device.

The message traffic from IO-Supervisors does not typically require real-time constraints. On the contrary, data exchange between IO-Controllers and IO-Devices require a temporal guarantee.

Since message communications in PROFINET IO networks are based on switched Ethernet, switching devices play an important role in handling all kinds of traffic (real-time and non real-time). The switching devices can be standard-Ethernet switches according to the IEEE 802.3 [IEE05], or modified ones. A simple PROFINET IO network connecting all types of PROFINET IO devices via one switching device is shown in Fig. 3.7.

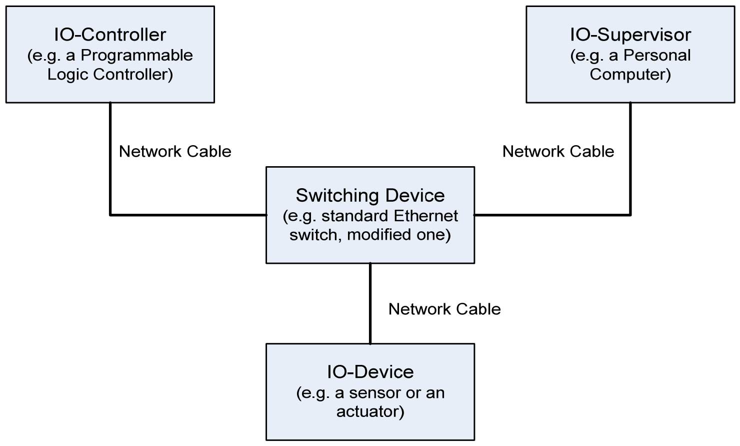

IO-Controller
(e.g. a Programmable Logic Controller)
IO-Supervisor
(e.g. a Personal Computer)
Network Cable
Network Cable
Switching Device
(e.g. standard Ethernet switch, modified one)
Network Cable
IO-Device
(e.g. a sensor or an actuator)

Fig. 3.7: A simple PROFINET IO network connecting all types of PROFINET IO devices

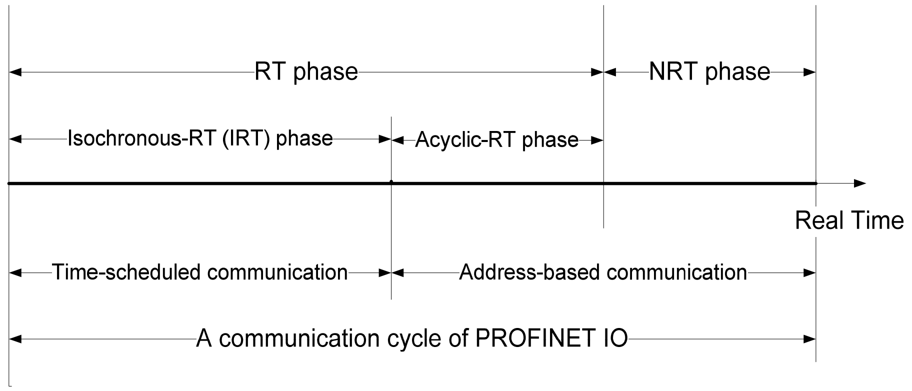

RT phase
NRT phase
Isochronous-RT (IRT) phase
Acyclic-RT phase
Real Time
Time-scheduled communication
Address-based communication
A communication cycle of PROFINET IO

Fig. 3.8: Traffic scheduling in one communication cycle of PROFINET IO

PROFINET IO uses the concept of TDMA (Time Division Multiple Access), where one communication cycle is divided into two phases: a real-time phase (RT phase) and a non real-time phase (NRT phase). This is depicted in Fig. 7.8. The RT phase contains two kinds of traffic: isochronous real-time (IRT) traffic and acyclic real-time (acyclic-RT) traffic. The IRT traffic is offline-scheduled and cyclically transmitted. The acyclic-RT traffic is triggered by an event occurrence, and transmitted with a real-time constraint. Acyclic-RT traffic is typically derived from an occurrence of an alarm event. The NRT phase used for NRT traffic occurs at the last period of a communication cycle, as depicted in Fig. 7.8. The NRT traffic may be standard TCP/IP, UDP/IP, raw Ethernet or PROFINET CBA traffic. A communication cycle of

PROFINET IO is continuously repeated by each station in a PROFINET IO network. The IRT phase, which is the beginning part of a communication cycle of PROFINET IO, needs precise clock synchronization. This denotes that all traffic of a PROFINET IO device can be transmitted into the underlying PROFINET IO network only if the precise clock synchronization of the device is established. In a PROFINET IO network, IRT frames are passed through the underlying switching devices without any interpretation of the address information in the IRT frames. The switching devices can do so since they contain a common predefined schedule. In the acyclic-RT phase, switching devices behave as standard Ethernet switches, where they relay acyclic-RT frames from one port to other ports with regard to the address information in the frames. All NRT frames are transmitted within the NRT phase on a first-come first-serve basis. All switching devices for PROFINET IO are synchronized by means of IEEE 1588 [IEE02] with the extension proposed by [JSW04], where accumulating local clock errors caused by cascading switching devices are alleviated.

The implementation of automation tasks may be subdivided in several application processes executed by different PROFINET IO devices. The data exchange of application processes between PROFINET IO devices exclusively takes place via Application Relationships (ARs). An AR contains all necessary data for establishment of data exchange. An AR consists of one or more Communication Relationships (CRs). In PROFINET IO there are three kinds of data exchange services in a CR: I/O Data Object, Alarm Data Object, and Record Data Object. These are described as follows:

- IO Data Object: I/O data objects are used for transmission of IRT traffic between an IO-Controller and an IO-Device. This transmission occurs within the isochronous phase of a communication cycle of PROFINET IO. IO data between I/O data objects is transferred via an IO Data CR, as shown in Fig. 3.9.
- Alarm Data Object: An alarm object transfers its alarm data when an alarm event occurs. It can send its alarm data only within the acyclic-RT phase of a communication cycle of PROFINET IO. The reception of an alarm object has to be explicitly acknowledged. Alarm data between alarm objects is transferred via an Alarm CR, as shown in Fig. 3.9.
- Record Data Object: Record data objects are used for configuration, monitoring, and diagnosis. A record data object transfers its record data via Record Data CRs. It does this within the NRT phase of a communication cycle. This is shown in Fig. 3.9.

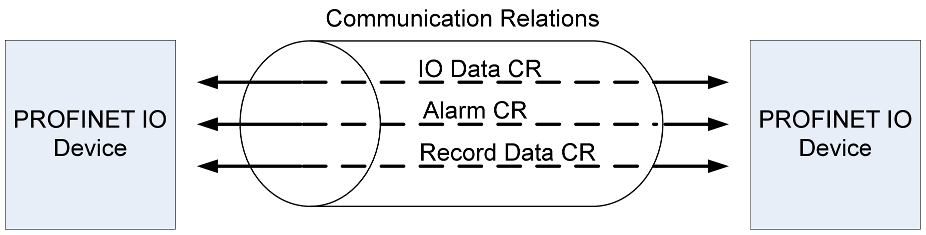


Fig. 3.9: Communication Relations (CRs) between two PROFINET IO devices

Similar to PROFIBUS DP [CSCM06], all PROFINET IO field devices must be described with a GSD (General Station Description) file [KDDS08], which is in XML format. A GSD file contains all technical information and functions. This information is relevant for engineering and data exchange. GSD files are provided by device manufacturers. A vendor-independent configuration tool reads information from a GSD file, and downloads it to an IO controller. The IO controller uses this information to configure and parameterize all its associated IO devices. This is carried out during runtime, and before entering the cyclic data exchange mode. IO-Devices enter the cyclic data exchange state after configuration and parameterization, if the startup state has been successful. The data exchange transmission of all frame types (Isochronous frames, acyclic-RT frames, and NRT frames) is performed in their own individual phase in a communication cycle. All RT, Isochronous and acyclic-RT frames are identified with a value of 0x8892 in the *EtherType* field, according to the standard Ethernet frame format. In addition, all RT frames bypass the UDP/IP stack, and directly access the Ethernet MAC layer.

3.2.2 Performance of PROFINET

PROFINET IO supports stringent timing constraints that are suitable for hard real-time applications, whereas PROFINET CBA does not. PROFINET IO can transport all kinds of traffic within the same PROFINET IO network. The RT communication in PROFINET IO follows a fixed communication schedule. Each communication cycle starts with the transmission of IRT process data. The duration of each cycle is defined as being between 31.25 and 4 milliseconds [Fel04]. Duration of the real-time phase is recommended not to exceed 60% of a communication cycle duration (50% for IRT frames, and 10% for acyclic-RT frames), so that the remaining time duration can be used for transmission of NRT traffic [PI09]. This implies that at least two maximum-sized standard Ethernet frames based on Fast Ethernet (100 Mbps) are able to be transmitted in 1 millisecond of a communication cycle [PI09]. This corresponds to a transmission time of approximately 250 microseconds. According to [Eth10a], there currently exists a PROFINET IO network implementation with the lowest communication cycle time of 0.5 milliseconds. To handle all kinds of traffic, standard-Ethernet switches, which are based on address-based switching, are modified as IRT suitable switches. IRT suitable switches can forward IRT traffic on a time synchronization-based switching basis. With regard to an IRT suitable switch, routes between receiving and transmitting ports are able to be predefined. The IRT communication of PROFINET IO can be done by employing the PTCP (Precision Transport Clock Protocol) of the IEEE 1588 standard. This is used to synchronize the local clocks of all stations and IRT suitable switches in a PROFINET IO network [KDDS08]. With PTCP, the effect to clock precision caused by many cascading switches in a largely distributed automation system is considerably reduced. To achieve very low jitter in the IRT communication of PROFINET IO, devices and IRT suitable switches require hardware implementations with ASIC (Application-Specific Integrated Circuit) technology. As a result, a jitter of below 1 microsecond can be achieved in IRT communication. In addition, IRT communication can achieve a low transport delay by introducing the cut-through technique of switched Ethernet in an IRT suitable switch [JE04] (e.g. HARTING Ha-VIS Fast Track Switch [HTG10]). However, the cut-through technique is not compliant with full-duplex Ethernet bridged technology, which uses the store-and-forward technique [JE04] [IEE98]. According to [JSW07] and [Pry08], a transport delay in a cut-through Ethernet switch is approximately 3 microseconds.

3.3 MODBUS-TCP

MODBUS-TCP [Acr05] [LL06b] [JH08] is one of several industrial Ethernet communication protocols developed by the Schneider Electric Company. MODBUS-TCP is a variant of the MODBUS protocol family, and is based on the TCP/IP protocol suite. MODBUS is a serial communication protocol for a fieldbus network. It connects field devices such as: sensors, actuators, and field device controllers such as PLCs (Programmable Logic Controllers). A MODBUS-TCP frame is embedded into a TCP frame in order to utilize available TCP/IP protocol networks, reducing development efforts when connecting fieldbus networks to office networks. However, MODBUS-TCP is not a timing-deterministic protocol.

3.3.1 Structure and Functionality of MODBUS-TCP

MODBUS-TCP is an application layer messaging protocol. A MODBUS-TCP frame, called an Application Data Unit (ADU), consists of a MODBUS Application Protocol (MBAP) header, and a Protocol Data Unit (PDU). This is shown in Fig. 3.10. The PDU is derived from a traditional MODBUS data frame, but without either the address or checksum fields. MODBUS-TCP employs the *Frame Check Sequence* field of the Ethernet frame format, instead of the checksum field of the more traditional MODBUS data frame format. The address field of the traditional MODBUS data frame format is not used in MODBUS-TCP. It is instead supplanted by the *Unit Identifier* field, as depicted in Fig. 3.10. Each ADU field is described as follows: The

Transaction Identifier field is used for synchronization between messages of transaction pairing. The *Protocol Identifier* field identifies protocol type. It is equal to zero for MODBUS-TCP. The *Length Field* contains the number of the remaining bytes in the remaining fields (the *Unit Identifier*, *Function Code*, and *Data* fields). The *Unit Identifier* field is used to identify a slave or server address.

MODBUS-TCP makes use of the TCP/IP standard model for MODBUS-TCP message transport, as shown in Table 3.2. MODBUS-TCP transports application data at the application layer, and employs TCP/IP and Ethernet to carry it between stations. The ADU of MODBUS-TCP is embedded into the data field of a standard TCP frame, as shown in Fig. 3.11. The TCP frame with the ADU is sent via port 502, which is specifically reserved for MODBUS-TCP applications. MODBUS-TCP shares the same physical and data link layers of standard Ethernet (IEEE 802.3 [IEE05]). This means that MODBUS-TCP is fully compatible with standard Ethernet devices.

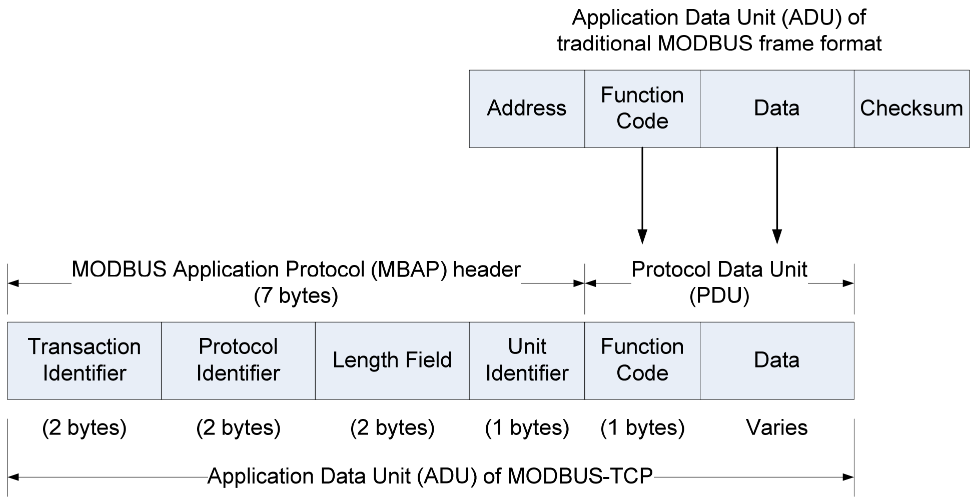


Fig. 3.10: Application Data Unit (ADU) of MODBUS-TCP

MODBUS-TCP is a connection-oriented protocol since it uses the Transmission Control Protocol (TCP). MODBUS-TCP follows a client/server model at the application layer, as shown in Fig. 3.12. This model is also referred to as the master/slave model (i.e. a MODBUS-TCP client and server can also be referred to as a MODBUS-TCP master and slave, respectively). With a client/server model, a MODBUS-TCP device (the client) sends a request message to another, required MODBUS-TCP device (the server). The server can send a response message when it has received the request message. This implies that only a client can initiate message transaction. However, some devices can operate as both clients and servers. If a server is unable to process a request from a client, it returns an exception response containing an error function code. All functions of MODBUS-TCP operate on memory registers to control, monitor, and configure I/O devices. A connection of MODBUS-TCP messages is a point-to-point communication path between two MODBUS-TCP devices. Each connection requires a source address, a destination address, and a connection ID, in each direction. Therefore, MODBUS-TCP communication is restricted to unicast messages. The MODBUS-TCP client may establish many simultaneous TCP connections at any given time. Each TCP connection uses a different, unused port to send its message. In the meantime, a MODBUS-TCP server receives all messages via port 502, which is used as a reserved listening/receiving port. In cases where a message is lost in transit, retransmission is required.

Table 3.2: MODBUS-TCP model, OSI reference model and TCP/IP standard model

Layer	OSI reference model	TCP/IP standard model	MODBUS-TCP model
7	Application	Application	Application (MODBUS-TCP)
6	Presentation		
5	Session		
4	Transport	Transport	Transport (TCP)
3	Network	Internet	Internet
2	Data Link	Data Link	Data Link (Ethernet)
1	Physical	Physical	Physical (Ethernet)

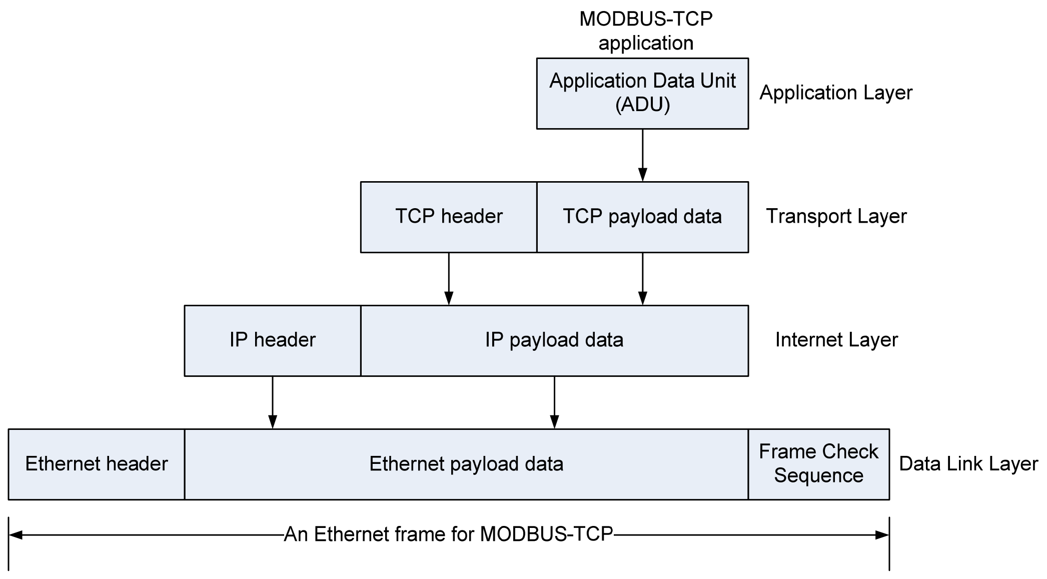


Fig. 3.11: Construction of an Ethernet frame for MODBUS-TCP

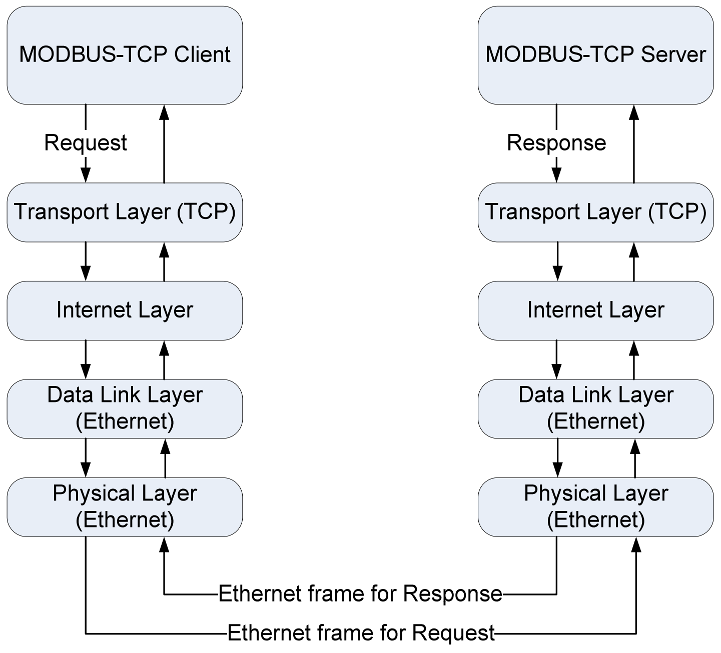

MODBUS-TCP Client
MODBUS-TCP Server
Request
Response
Transport Layer (TCP)
Transport Layer (TCP)
Internet Layer
Internet Layer
Data Link Layer (Ethernet)
Data Link Layer (Ethernet)
Physical Layer (Ethernet)
Physical Layer (Ethernet)
Ethernet frame for Response
Ethernet frame for Request

Fig. 3.12: The Client/Server communication model of MODBUS-TCP

3.3.2 Performance of MODBUS-TCP

Since MODBUS-TCP transports data over the TCP/IP protocol, MODBUS-TCP performance is very similar to TCP/IP performance. MODBUS-TCP provides non-deterministic communication, the same as TCP/IP and Ethernet. This implies that MODBUS-TCP is not suitable for hard real-time applications with low latency requirements. Like Ethernet, MODBUS-TCP can be implemented in every type of topology (line, ring, or star). Therefore, the transport delay of MODBUS-TCP messages between devices depends on network topology and hardware. Although MODBUS-TCP is not suitable for high-speed and hard real-time applications, it is adequate to use in low-speed and soft real-time applications. MODBUS-TCP also works well with non-control systems where high critical-time is not the case (e.g. for maintenance and diagnosis purposes). With regard to the client/server communication model, the response time of a MODBUS-TCP transaction is taken into account for real-time constraints. The timeout mechanism at a MODBUS-TCP client for MODBUS-TCP message retransmission should not be smaller than the expected maximum response time. If the client timeout is larger than the expected maximum response time, message congestion will occur in the underlying network. The client timeout is dependent on several factors, such as: network topology, bandwidth of network interface hardware, and the amount of other traffic within the network.

3.4 SERCOS III

SERCOS III [SER07] was designed to merge the hard real-time aspects of the SERCOS interfaces (SERCOS I and SERCOS II) with Ethernet. SERCOS I and II are standardized open digital interfaces for communication between industrial controls, motion devices (drives), and input/output (I/O) devices. Modern Ethernet bandwidth is extremely high, but Ethernet is not a deterministic protocol. SERCOS III provides deterministic communication behavior to standard Ethernet, while maintaining the important features of the SERCOS I and II interfaces (e.g. topology, profiles, telegram structure, and synchronization). Additionally, SERCOS III supports real time traffic coexistence with standard-Ethernet traffic.

3.4.1 Structure and Functionality of SERCOS III

SERCOS III uses a master/slave communication structure. A SERCOS III network consists of one master device and one or more slave devices. Both master and slave devices (all SERCOS III devices) have two ports each, as depicted in Fig. 3.13. Both ports of a SERCOS III device are standard-Ethernet ports (IEEE 802.3 [IEE05]). The network topologies supported by SERCOS III are ring and line topologies, as illustrated in Fig. 3.13. A master device in both topologies is responsible for starting the transmissions of telegrams. A slave device is responsible for relaying telegrams to the next slave device. A slave device returns telegrams back to the previously transmitting slave device if there is no further slave device. In a ring topology network, two ports of a master device are used and looped via slave devices. Note that SERCOS III does not use the star topology seen in switched Ethernet. This means that no hub or switch is needed in a SERCOS III network. Therefore, there is no delay time or jitter caused by Ethernet switches or hubs. However, the hardware development of SERCOS III requires considerable effort for both topologies. With regard to ring topology as a SERCOS III network structure, a master device initiates two identical telegrams. Each of these telegrams is transmitted via a different port of the master device. Each slave device relays an incoming telegram to the next slave device, as depicted in Fig. 3.13 (b). The last slave device which connects to another port of the master device forwards the telegram back to the master device. The flows of two identical telegrams shown in Fig. 3.13 (b) follow the sequence numbers 11→12→13→14 and 21→22→23→24. Note that the two identical telegrams transmitted from the master device arrive approximately at the same point in time. In cases where a line topology is used as a SERCOS III network structure, the last slave device returns an incoming telegram back to the previously sending device. The telegram then flows back via the same route as it was transported, as depicted in Fig. 3.13 (a) (i.e. the telegram flow is 1→2→3→4→5→6). SERCOS III has a fault tolerance feature in a ring network structure. If a break at a network cable takes place at any point in a ring network structure, the system automatically switches over from a ring structure to a line structure, as illustrated in Fig. 3.13 (b) and Fig. 3.14. In Fig. 3.14, two new flows for two identical messages from the master device are 11→12, and 21→22→23→24. A SERCOS III network in line topology does not offer a redundant telegram transfer, but less cabling than for a ring network is typically required.

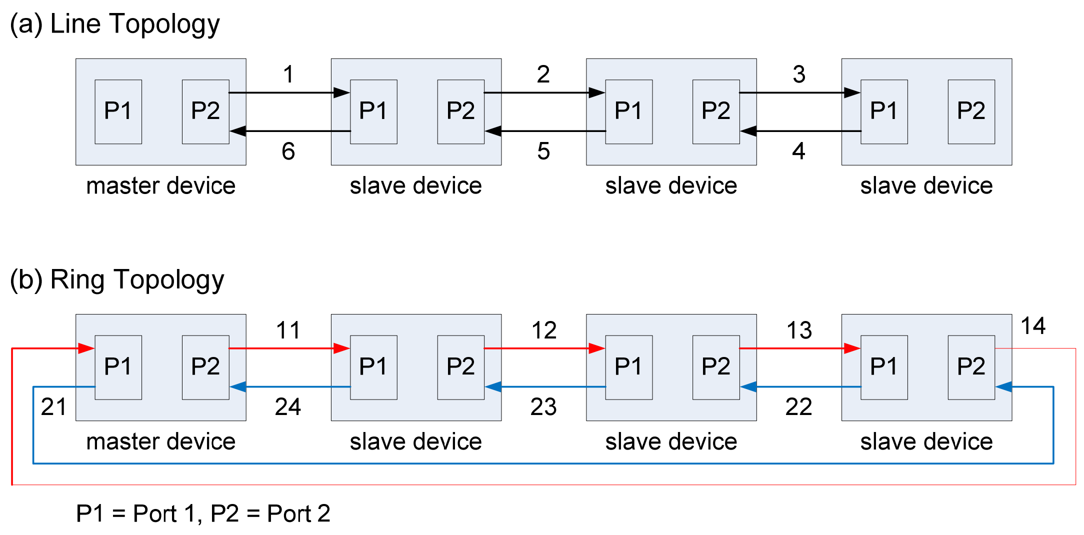


Fig. 3.13: Two SERCOS III network structures: (a) Line Topology, (b) Ring Topology

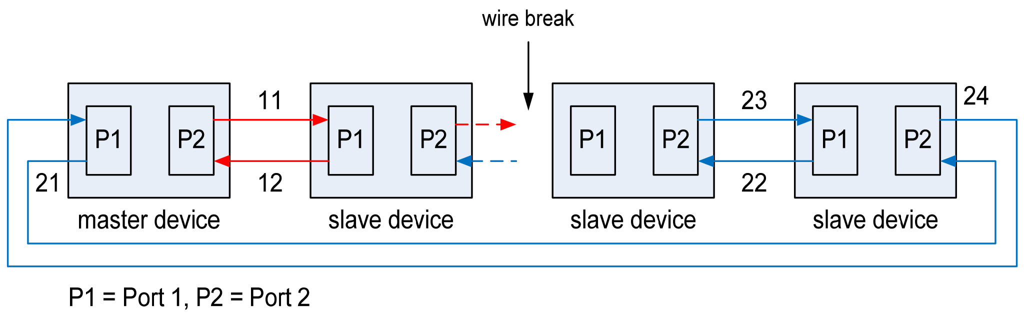


Fig. 3.14: Two message flows in a ring network structure of SERCOS III when a wire breaks

SERCOS III supports both real-time and non real-time traffic in the separated channels of one communication cycle (real-time traffic in the real-time channel, non real-time traffic in the non real-time channel). This is depicted in Fig. 3.15. A communication cycle time of SERCOS III ranging from 31.25 microseconds to 65 milliseconds is chosen by the user depending on the given application. The real-time channel is divided into two parts with respect to telegram type, these are the Master Data Telegrams (MDTs) and Answer Telegrams (ATs), as shown in Fig. 15. SERCOS III supports up to 4 MDT telegrams and 4 AT telegrams in a communication cycle. MDT telegrams contain information (command data), whereas AT telegrams contain status data (e.g. feedback values and sensor data). Slave devices fill status data in predefined areas in AT telegrams, update the checksum, and then pass the AT telegrams to the next device. The status data in AT telegrams is read by the master device or other slave devices. SERCOS III nodes allow the non real-time channel of one communication cycle to be used to exchange non real-time data in other Ethernet-based messages, such as TCP/IP messages.

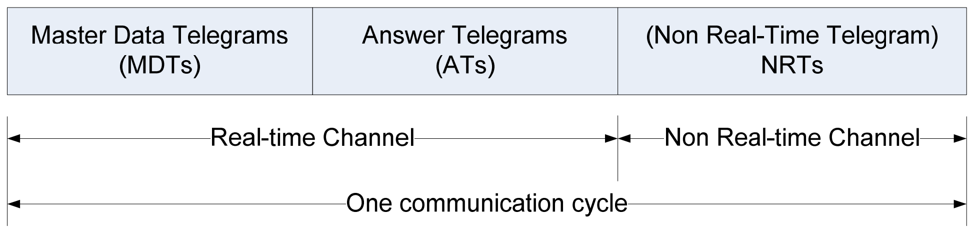


Fig. 3.15: One communication cycle of SERCOS III

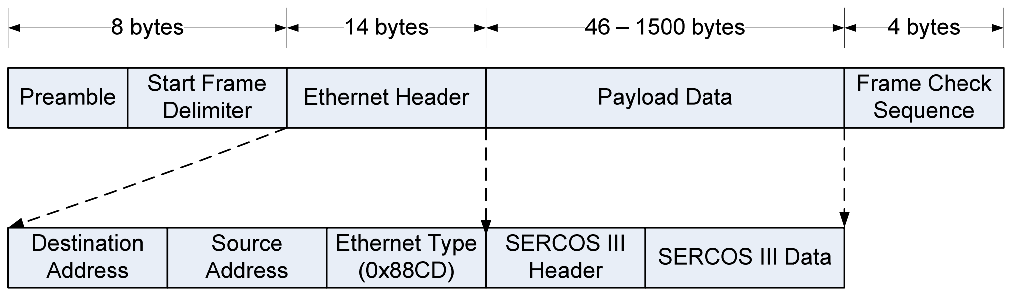


Fig. 3.16: The structure of a SERCOS III telegram

A SERCOS III telegram is based on the standard-Ethernet frame format. The *payload data* field of the standard-Ethernet frame format contains the two main fields of a SERCOS III telegram: SERCOS III header and SERCOS III data, as shown in Fig. 3.16. The *Ethernet type* field, with a unique value of 0x88CD, identifies SERCOS III data. All SERCOS III telegrams are issued by a master device and intended for all slave devices. Therefore, the *Destination Address* field of the standard-Ethernet frame format is defined as a broadcast address (all ones). The *Source Address* field is assigned with the MAC address (Ethernet hardware address) of a master device. The *SERCOS III header* field contains control and status information specific to SERCOS III. The *SERCOS III data* field carries values in a set of variables (e.g. feedback values or set-point values) in each device within a SERCOS III network.

With regard to synchronization, SERCOS III uses the first MDT telegram in a communication cycle for synchronizing the local clocks of slave devices with the local clock of their master device. The Master Synchronization Telegram (MST) field in the first MDT telegram is used for this synchronization purpose. Precise clock synchronization in SERCOS III networks can be achieved without the timing information of local clocks. A synchronization interval of SERCOS III is a time interval between the MST fields of two consecutive first MDT telegrams, as depicted in Fig. 3.17. This synchronization interval is an exact equidistant time interval and equal to a communication cycle time.

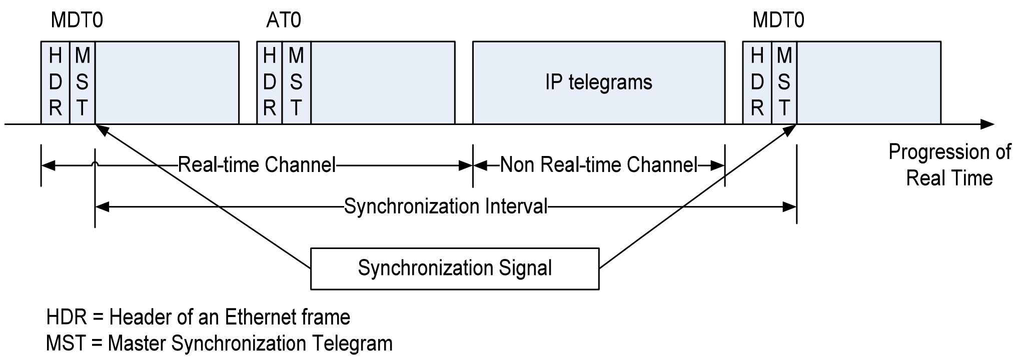


Fig. 3.17: Synchronization interval in SERCOS III

Unlike a master/slave communication (e.g. Ethernet POWERLINK communication), slave devices in a SERCOS III network can communicate with other slave devices without waiting for control messages from the master device each time. Communication between multiple slave devices in a SERCOS III network is called *Cross Communication* (CC), as shown in Fig. 3.18. Slave devices in a SERCOS III network can also communicate with other slave devices in other SERCOS III networks, via the master device of each network. Communication between master devices is called *Controller to Controller* (C2C) communication, as shown in Fig. 3.18.

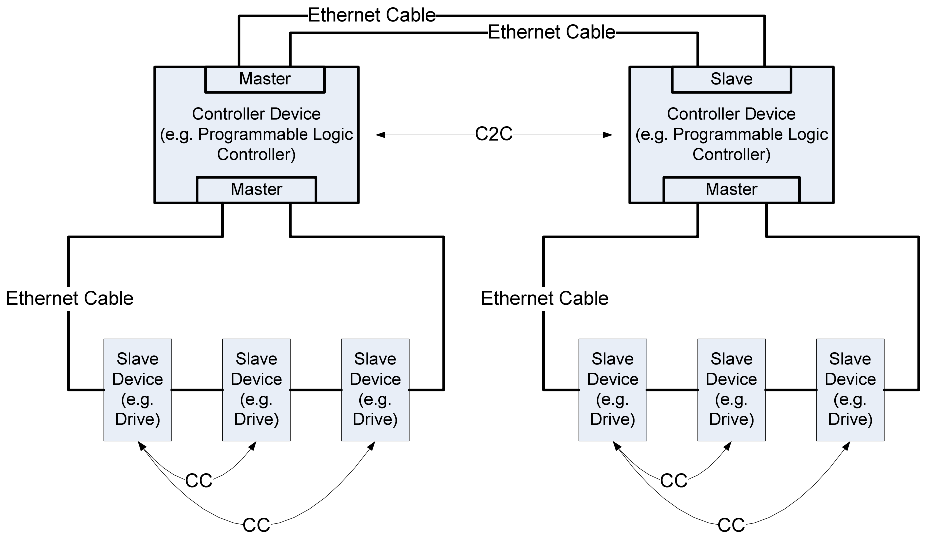

Ethernet Cable
Ethernet Cable
Master
Slave
Controller Device (e.g. Programmable Logic Controller)
C2C
Controller Device (e.g. Programmable Logic Controller)
Master
Master
Ethernet Cable
Ethernet Cable
Slave Device (e.g. Drive)
Slave Device (e.g. Drive)
Slave Device (e.g. Drive)
Slave Device (e.g. Drive)
Slave Device (e.g. Drive)
Slave Device (e.g. Drive)
CC
CC
CC
CC

Fig. 3.18: Cross Communication (CC) and Controller to Controller (C2C) in two ring networks

NRT frames based on standard Ethernet (e.g. TCP/IP messages from outside networks) can access SERCOS III networks in two methods. The first method is to make use of unused ports of SERCOS III devices, as depicted in Fig. 19 (a). This figure illustrates the NRT frames from a laptop accessing a SERCOS III network via the unused SERCOS III port of the last slave device. The second method is to use a SERCOS III device with one or more additional Ethernet ports installed, as shown in Fig. 19 (b). With such a device, NRT frames can access anywhere along a SERCOS III network. Such Commercially these special devices are available commercially (e.g. netSwitch SERCOS III [Hil08]).

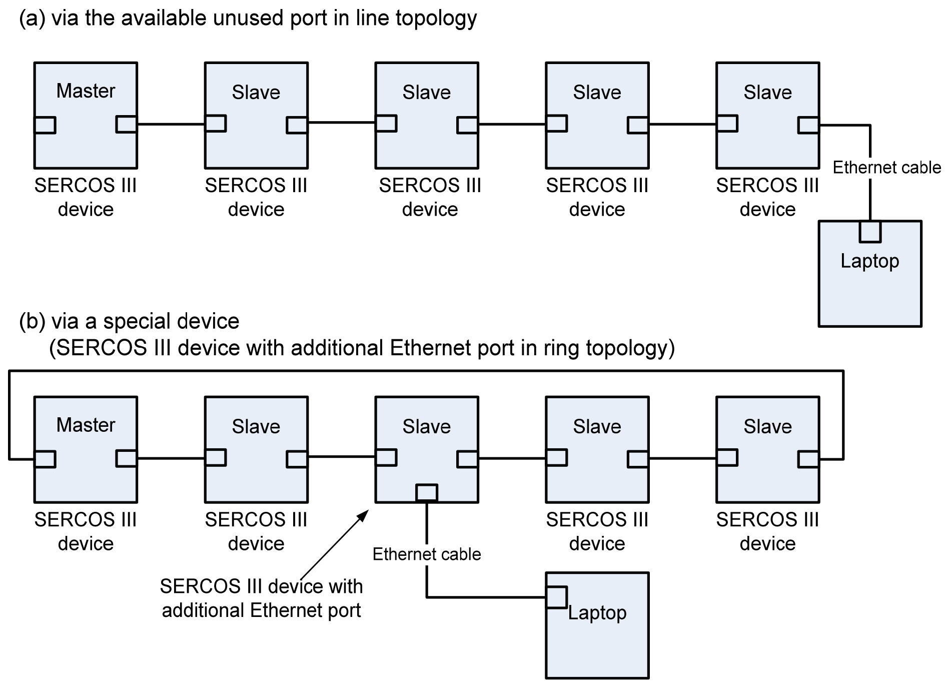


Fig. 3.19: NRT-frame access (a) via an available unused port (b) via a custom device

3.4.2 Performance of SERCOS III

Based on modern Ethernet technology, SERCOS III can obtain higher transmission speeds than SERCOS I and SERCOS II. SERCOS III uses hardware-based synchronization to achieve a jitter of less than 1 microsecond. With such a small jitter, SERCOS III is a deterministic protocol and allows links to the existing manufacturing communication infrastructure. Non real-time traffic based on Ethernet is allowed to coincide with SERCOS III real-time traffic in a SERCOS III network. Non real-time messages can only be transported within the non real-time channel of a communication cycle. The SERCOS I and SERCOS II interfaces use fiber optic interfaces for noise immunity. With modern Ethernet technology, SERCOS III can make use of twisted-pair cables without noise problems. SERCOS III also has a fiber optic-based implementation for high-noise-critical applications. Regarding fault tolerance, all SERCOS III devices in a ring network structure can continue operation if a wire break occurs. SERCOS III also supports a hot plugging feature, where a SERCOS III device can be inserted and removed during network operation. SERCOS III boasts highly efficient use of Fast Ethernet bandwidth (i.e. up to 1,494 bytes from all SERCOS III devices in one Ethernet frame are bundled together with 44 bytes of overhead data). Therefore, the bandwidth utilization increases up to 97% for Fast Ethernet. However, transport delay between SERCOS III devices depends on all interconnected nodes used in each transmission. This is because all SERCOS III telegrams must travel through each interconnected node.

3.5 EtherCAT

EtherCAT [Eth09] [Eth10b] stands for **E**thernet for **C**ontrol **A**utomation **T**echnology. It was designed to target high-performance Ethernet-based fieldbus systems requiring a short cycle time, and a low communication jitter. Typical data length per node in a fieldbus network is very small, and less than the minimum length (46 bytes) of the payload data in an Ethernet frame. EtherCAT increases Ethernet bandwidth utilization by all nodes sharing Ethernet frames, rather than generating individual Ethernet frames. EtherCAT utilizes a standard-Ethernet device for generating standard-Ethernet frames for a whole EtherCAT network. Each of the other devices in an EtherCAT network requires a dedicated hardware device instead of a standard-Ethernet device. EtherCAT was integrated into the international fieldbus standard IEC 61158 [IEC00] and IEC 61784-2 [IEC07].

3.5.1 Structure and Functionality of EtherCAT

An EtherCAT network structure consists of one node for the master (EtherCAT-master node) and one or more nodes for the slaves (EtherCAT-slave nodes). An EtherCAT-master node uses a standard-Ethernet medium access controller (MAC) for generating standard-Ethernet based frames. An EtherCAT-slave node requires dedicated hardware for an EtherCAT network interface. This dedicated hardware comprises two physical Ethernet ports that support bi-directional communication. Fig. 3.20 depicts an EtherCAT network in which an EtherCAT-master node connects to three EtherCAT-slave nodes in a line topology. The EtherCAT-master node is responsible for initiating EtherCAT messages, and transmitting them to the EtherCAT-slave node (a). Each EtherCAT-slave node passes the EtherCAT messages from one port to another. In other words, the EtherCAT messages are not received and subsequently retransmitted by any of the EtherCAT-slave nodes. While the EtherCAT messages are crossing from one port to another at one of the Ethernet-slave nodes, the node extracts the output data addressed to it and inserts the new input data *on-the-fly*. The last EtherCAT-slave node (node (c) in Fig. 3.20) returns the EtherCAT message back to the EtherCAT-master node. Notice that the physical connection in an EtherCAT network is a line structure, but that communication occurs in a logical ring structure. In Fig. 3.20, the last EtherCAT-slave node in the row (c) detects no connection to other EtherCAT-slave nodes, and then automatically short circuits a pair of transceivers (Rx and Tx) together. Therefore, EtherCAT messages always loop back to the EtherCAT-master node at the last EtherCAT-slave node.

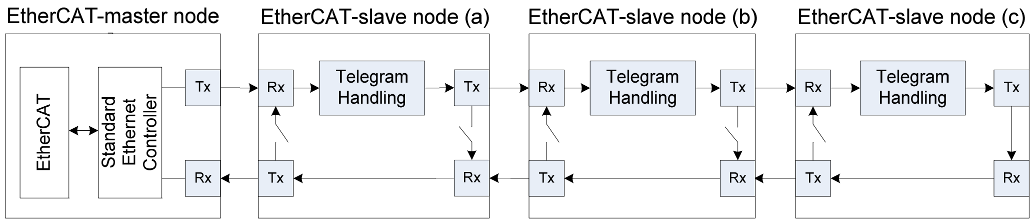


Fig. 3.20: An Ethernet network in a physical line structure

A physical network structure of EtherCAT can be extended in the form of a branch. A branch extension in an Ethernet network involves adding a network interface in an EtherCAT-slave node. Fig. 3.21 shows the EtherCAT-slave node (a) with three network interfaces making a branch connection with another EtherCAT-slave node (d).

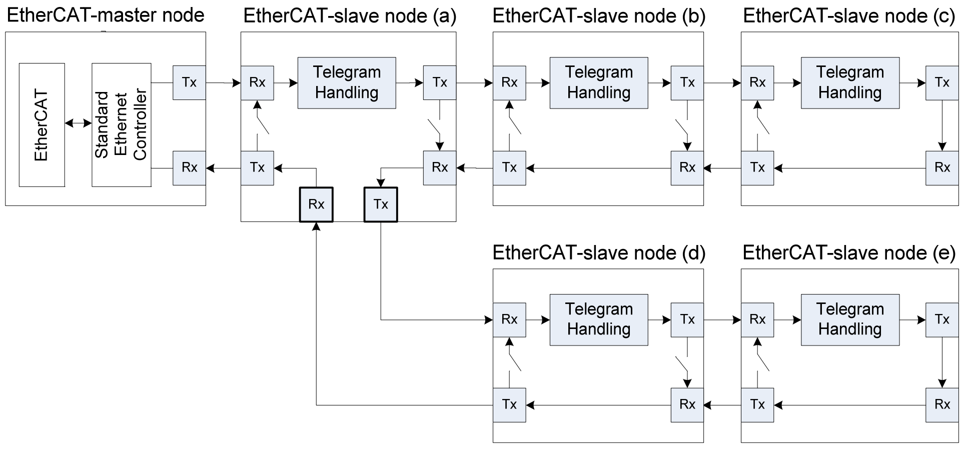


Fig. 3.21: A branch extension in an EtherCAT network

EtherCAT uses a single large Ethernet frame for carrying EtherCAT telegrams to and from each EtherCAT device. There are two structures an EtherCAT message can take: the first is based on the standard-Ethernet frame format, while the second is based on the UDP (User Datagram Protocol) message format. These are shown in Fig. 3.22 (a) and Fig. 3.22 (b), respectively. When an EtherCAT message is based on the standard-Ethernet frame format, the EtherCAT header and EtherCAT telegrams are embedded in the *payload data* field of the standard-Ethernet frame, as depicted in Fig. 3.22 (b). EtherCAT telegrams are used for exchanging real-time data among an EtherCAT-master node and EtherCAT-slave nodes. EtherCAT messages based on the standard-Ethernet frame format are used in exclusive EtherCAT networks. EtherCAT networks can also be connected with other networks, e.g. office networks or even additional EtherCAT ones. This is achieved using an EtherCAT message based on the UDP frame format. The EtherCAT header and all EtherCAT telegrams based on UDP are encapsulated within a UDP frame, as depicted in Fig. 3.22 (b).

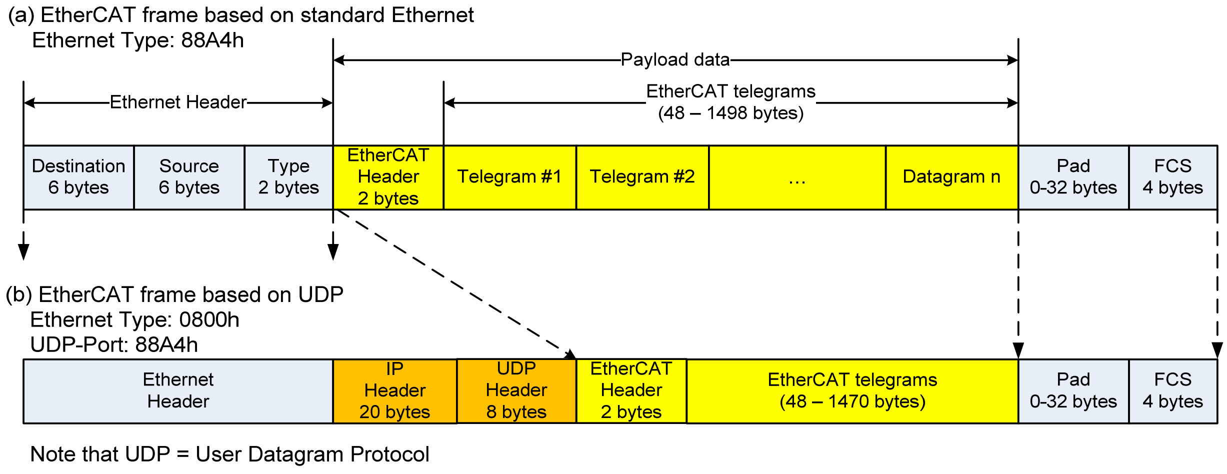


Fig. 3.22: An EtherCAT message based on standard Ethernet frame (a), and UDP (b)

An EtherCAT frame consists of one EtherCAT header and several EtherCAT telegrams, as shown in Fig. 3.23. An EtherCAT telegram begins with a 10-byte telegram header, which specifies the type of operation (e.g. read or write). The data field of an EtherCAT telegram contains process data, which is distributed among EtherCAT devices. The last field of an EtherCAT telegram, called the *working counter*, is used by an EtherCAT-master node to check whether an EtherCAT telegram was correctly processed at each addressed EtherCAT-slave node. During the operation of extracting and/or inserting process data to an EtherCAT telegram, each addressed EtherCAT-slave node increments the working counter's value. The EtherCAT-master node, knowing how many EtherCAT-slave nodes are addressed by an EtherCAT telegram, can check from its *working counter* whether all EtherCAT-slave nodes have exchanged their data. EtherCAT can also check whether EtherCAT telegrams have been correctly transmitted, by verifying the checksum value in the *frame check sequence* (FCS) field of an Ethernet frame.

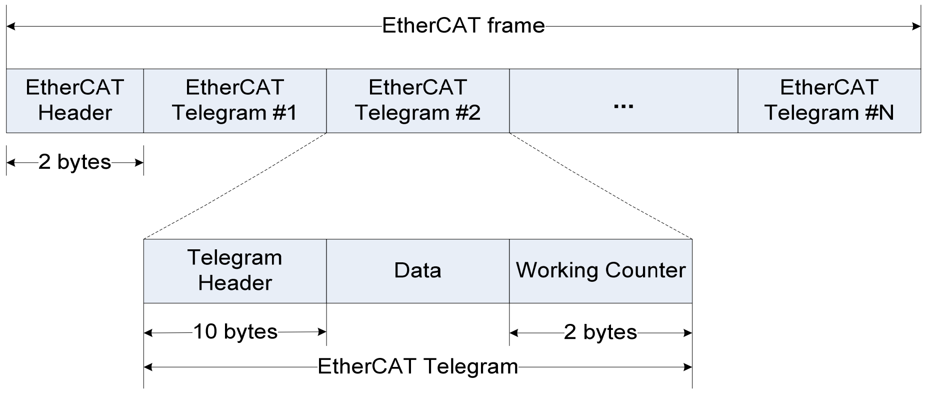


Fig. 3.23: The structure of an EtherCAT frame and an EtherCAT telegram

EtherCAT supports a system availability property. This is implemented by connecting the last EtherCAT-slave node to an additional standard Ethernet port on the EtherCAT-master node, as depicted in Fig. 3.24. This turns the physical line structure into a physical ring structure. A ring structure guarantees that if any EtherCAT-slave device or cable fails, then the functional EtherCAT-slave devices can reroute traffic such that the system continues to operate. This is illustrated in Fig. 3.25. Note that the reroute of traffic occurs in one cycle time.

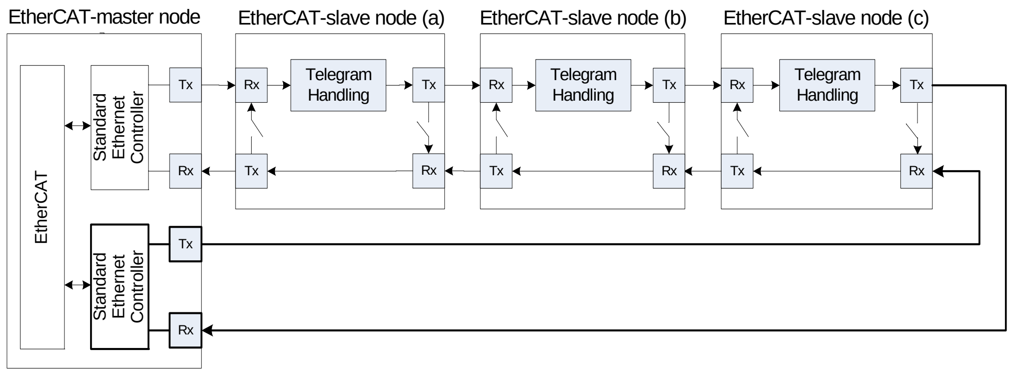


Fig. 3.24: An EtherCAT network in a physical ring structure

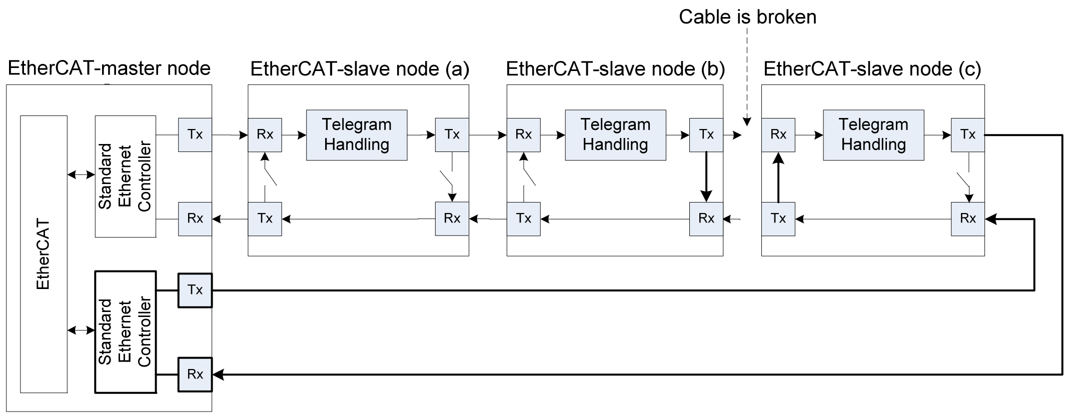


Fig. 3.25: A broken cable in a ring network structure of EtherCAT

EtherCAT does not use a notion of system-wide global-time for real-time communication. However, clock synchronization is required for simultaneous actions in some distributed applications (e.g. an application where several servo axes carry out their coordinated movements simultaneously). The clock synchronization of EtherCAT is fully based on hardware implementation. Each EtherCAT device measures the time difference between an incoming EtherCAT frame and its returning EtherCAT frames. With this time difference, an EtherCAT-master node can determine the offsets of the propagation delays between the EtherCAT-master node clock and each EtherCAT-slave clock. This implies that the EtherCAT-master node knows its local clock’s point in time when an incoming EtherCAT messages arrives at each EtherCAT-slave node. EtherCAT can achieve clock synchronization with a jitter of less than 1 microsecond. External synchronization (e.g. clock synchronization between several EtherCAT-master clocks) is based on the IEEE 1588 standard [IEE02].

3.5.2 Performance of EtherCAT

According to [Eth10b], EtherCAT achieves a cycle time of 300 microseconds, provided that up to 1486 bytes of process data for 12000 digital input and outputs are exchanged in a single Ethernet frame. In comparison to other industrial Ethernets (SERCOS III, Profinet-IRT, Ethernet POWERLINK and Profinet), EtherCAT takes the least cycle time (276 microseconds) under these conditions [Eth09]. With such a small cycle time, EtherCAT utilizes around 44% of total network bandwidth (i.e. 56% still remains for other traffic). However, only the EtherCAT-master node is built from COTS (commercial-of-the-shelf) components, and its functionality relies on software. On the contrary to EtherCAT-master nodes, all EtherCAT-slave nodes are implemented using dedicated hardware. Such dedicated hardware is not fully compatible with standard-Ethernet hardware. Additionally, EtherCAT cannot take advantage of future developments of standard Ethernet without a reimplementation of slave node hardware.

3.6 Ethernet POWERLINK

Ethernet POWERLINK [EPS08] is a *real-time Ethernet* protocol for industrial networks. It was first developed by the Austrian automation company (B&R Industrial Automation Corporation) as proprietary technology in November 2001. It was then released as an open technology in 2002. The Ethernet POWERLINK Standardization Group (EPSG) was formed in 2003, to manage the Ethernet POWERLINK standard. Later, Ethernet POWERLINK version 2 was made, which is fully integrated with the mechanisms of CANopen [BCF98].

3.6.1 Structure and Functionality of Ethernet POWERLINK

Ethernet POWERLINK was designed to be suitable for bus-based Ethernet (so-called shared Ethernet). With modern (switched) Ethernet using a star topology, Ethernet POWERLINK can be applied where all nodes connect to each other via standard-Ethernet switches. However, a network structure in star topology is not recommended for Ethernet POWERLINK, because it introduces high transport delays. Ethernet POWERLINK uses a *time-slot based master-slave communication* scheme. An Ethernet POWERLINK network consists of a master node called *Managing Node* (MN), and slave nodes called *Controller Nodes* (CNs). The data exchange between the MN and CNs is performed in predefined time slots. A CN can transmit frames as soon as it has received a request frame from the MN. The MN is responsible for organizing all cycle timings for the CNs. The MN starts a communication cycle by transmitting to all CNs a special, *Start of Cycle* frame (SoC). The SoC frames from the MN are used for synchronizing all CNs. After a SoC frame has been sent by the MN, the MN sends a Poll Request (PReq) frame to all CNs. In order to avoid communication conflict, only one of the CNs identified by the PReq frame can reply to a Poll Respond (PRes) frame. The MN continues sending other PReqs frames and then gets the corresponding PRes frames from the addressed CNs.

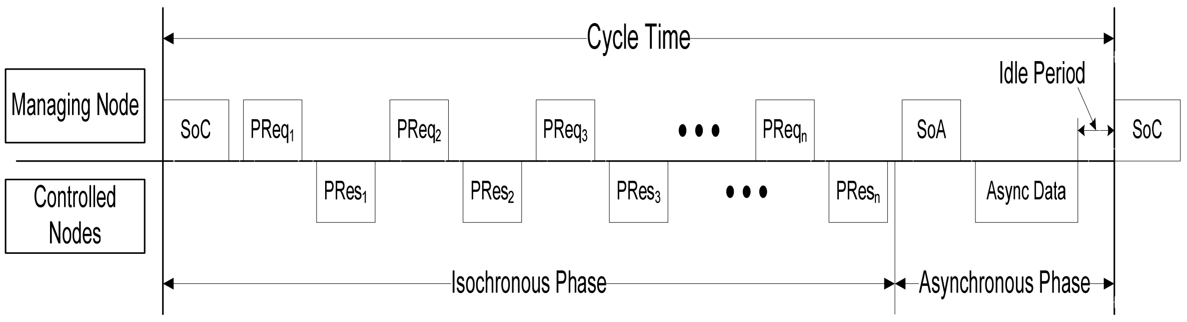


Fig. 3.26: Master-slave communication in one Ethernet POWERLINK cycle

One Ethernet POWERLINK cycle is divided into two phases: the isochronous phase and the asynchronous phase, as shown in Fig. 3.26. The first period of the cycle is the isochronous phase, in which all time-critical data is transmitted using PReq and PRes messages. The latter period of the cycle is the asynchronous phase, in which non real-time messages based on standard Ethernet (e.g. TCP/IP messages) are arbitrarily sent. After the isochronous phase, the MN informs all CNs of the beginning of the asynchronous phase by sending a *Start of Asynchronous* (SoA) frame to all CNs. Only one device of the controlled nodes is granted permission from the managing node to send non real-time messages in the asynchronous phase. All nodes are idle during the idle period of the asynchronous phase. The MN repeats the processes of the synchronous phase in the upcoming cycle.

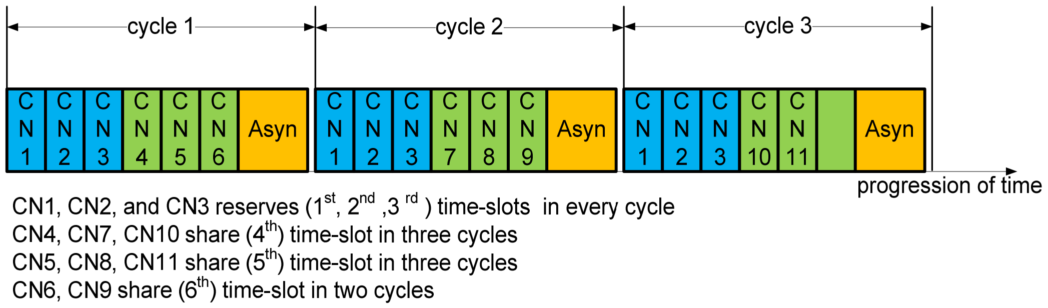

cycle 1
cycle 2
cycle 3
CN 1
CN 2
CN 3
CN 4
CN 5
CN 6
Asyn
CN 1
CN 2
CN 3
CN 7
CN 8
CN 9
Asyn
CN 1
CN 2
CN 3
CN 10
CN 11
Asyn
progression of time
CN1, CN2, and CN3 reserves (1st, 2nd ,3 rd) time-slots in every cycle
CN4, CN7, CN10 share (4th) time-slot in three cycles
CN5, CN8, CN11 share (5th) time-slot in three cycles
CN6, CN9 share (6th) time-slot in two cycles

Fig. 3.27: Time-slot sharing in Ethernet POWERLINK cycles

In order to enhance the bandwidth utilization of Ethernet POWERLINK, CNs can share time-slots within the isochronous phase, as illustrated in Fig. 3.27. In this figure there are three controlled nodes (CN1, CN2 and CN3) which have the same cyclic message period (one cycle time), while the other eight controlled nodes (CN4, CN5, CN6, CN7, CN8, CN9, CN10 and CN11) require a cyclic message period of three cycles. The eight controlled nodes can share their time-slots in different cycles (e.g. the CN4, CN7 and CN10 nodes share the same time slot but transmit in different cycles). Likewise the CN5, CN8 and CN11 nodes are assigned to use the same time-slot but in different cycles. Again, the CN6 and CN9 nodes are allowed to transmit in the same time-slot, but in different cycles.

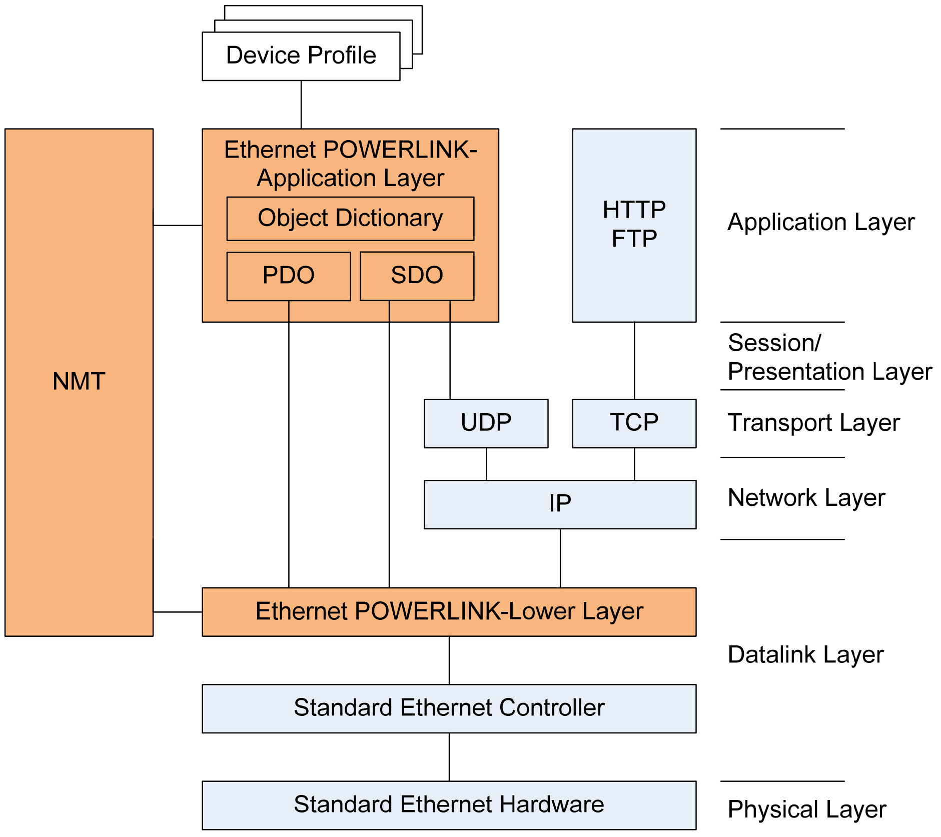

Device Profile
Ethernet POWERLINK-Application Layer
Object Dictionary
PDO
SDO
NMT
HTTP FTP
UDP
TCP
IP
Ethernet POWERLINK-Lower Layer
Standard Ethernet Controller
Standard Ethernet Hardware
Application Layer
Session/ Presentation Layer
Transport Layer
Network Layer
Datalink Layer
Physical Layer

Fig. 3.28: *Ethernet POWERLINK* protocol stack with reference to OSI Model [OSI96]

Ethernet POWERLINK runs on top of standard Ethernet. In Fig. 3.28, the Ethernet POWERLINK-Lower Layer is capable of accessing and handling standard-Ethernet controllers. The Ethernet POWERLINK-Lower Layer also follows a specified cycle schedule for cyclic data exchange. Ethernet POWERLINK transfers time-critical data via *Process Data Objects* (PDOs) in the isochronous phase of a communication cycle. Non time-critical data (e.g. diagnostic data and configuration parameters) are transported via *Service Data Objects* (SDOs) in the asynchronous phase. The SDO data transfer is based not only on exclusive Ethernet frames, but also on the UDP (User Datagram Protocol) of the TCP/IP protocol suite. However, a standard-Ethernet device without an Ethernet POWERLINK protocol stack cannot work in an Ethernet POWERLINK network. Standard Ethernet devices can be integrated in an Ethernet POWERLINK network using a dedicated gateway device, e.g. EPLGW (Ethernet to Ethernet POWERLINK Gateway) [SHF06]. The application layer of Ethernet POWERLINK adopts the *object dictionary* concept of CANopen [BCF98]. The object dictionary is an interface between the application and the communication. Each object dictionary entry directly allocates a reference to a variable containing application data. The communication services (PDOs and SDOs) access these application data variables directly. Therefore, migration from existing CANopen applications to Ethernet POWERLINK devices requires only trivial modifications.

3.6.2 Performance of Ethernet POWERLINK

Ethernet POWERLINK communicates cyclic process data during the isochronous phase of a communication cycle. A managing node (MN) uses a polling method to send a request frame, and then waits for the corresponding response frame from the targeted controlled node (CN). With such a polling method, Ethernet POWERLINK has to ensure that the fixed timeout interval (T_O), which starts when the transmission of a request frame begins, does not expire before the corresponding response frame from the targeted CN is received. In cases where the response frame from the targeted CN arrives late, the next request frame from the managing node may collide with the incoming response frame. Therefore, the duration of an Ethernet POWERLINK communication cycle depends on the response time of each Ethernet POWERLINK device. Since Ethernet POWERLINK was designed to be based on a bus network topology, the maximum transport delay in bus topology is lower than in star topology (because no transport delay occurs in switching devices). Ethernet POWERLINK can also be used in star network topology, but this is not recommended. In addition, the total number of CNs in an Ethernet POWERLINK network affects the length of the communication cycle. The higher the total number of controlled nodes, the larger the achievable cycle time is. According to [FS07], Ethernet POWERLINK achieves a jitter of less than 1 microsecond. A software implementation of the Ethernet POWERLINK-Lower Layer in typical micro-controllers can achieve a response time in the range of 3-10 microseconds [FS07]. The problem of using a software-based Ethernet POWERLINK-Lower Layer is the uncertainty of the response times when changing the hardware micro-controller platform. Instead of hardware micro-controller platforms, a dedicated hardware device (e.g. FPGA - Field Programmable Gate Array) is used, guaranteeing response times of less than 1 microsecond [FS07]. By using theoretical analysis [CSV08], the time duration of the isochronous phase for an Ethernet POWERLINK network with 16 controlled nodes and one Ethernet hub is 423.4 microseconds, regardless of propagation delays from network cables. If there are more than 16 controlled nodes under the same condition, the cycle time will be larger than 423.4 microseconds.

3.7 AFDX

AFDX (Avionic Full Duplex Switch Ethernet) [ARI06] [Con05] is a standard that defines the electrical and protocol specifications for the exchange of data between avionics subsystems. AFDX is contained in ARINC 664 Part 7 [ARI06], and was formally issued on June 27 2005. It was originally introduced by Airbus, and is based on modern Ethernet technology (Fast and Gigabit Ethernet). In comparison to its predecessors, it has a much higher bandwidth (up to 1000 Mbps compared to the 2 Mbps supported by ARINC 629 [ARI99]). AFDX utilizes COTS Ethernet devices, and extends standard Ethernet with the bandwidth allocation mechanism for AFDX traffic. AFDX provides high data integrity via the AFDX frame format, and supports fault-tolerant operation by redundant message communication.

3.7.1 Structure and Functionality

Avionics subsystems connecting components of a control system (e.g. sensors, actuators, and controllers) communicate with each other via an AFDX network, as depicted in Fig. 3.29. The AFDX network, which is structured based on a star network topology, consists of three main elements: an AFDX End System, an AFDX Switch and an AFDX Link, as depicted in Fig. 3.29. The End System is an interface between an Avionics Subsystem and an AFDX Switch (via an AFDX Link). The AFDX Link makes use of a standard Ethernet cable providing full-duplex communication. An Avionics computer system comprises one or more Avionics subsystems and one End System, as depicted in Fig. 3.29. According to ARINC 653 [ARI05], an interface of a partition-based operating system is required to communicate between Avionics subsystems, so that any faults occurring do not propagate.

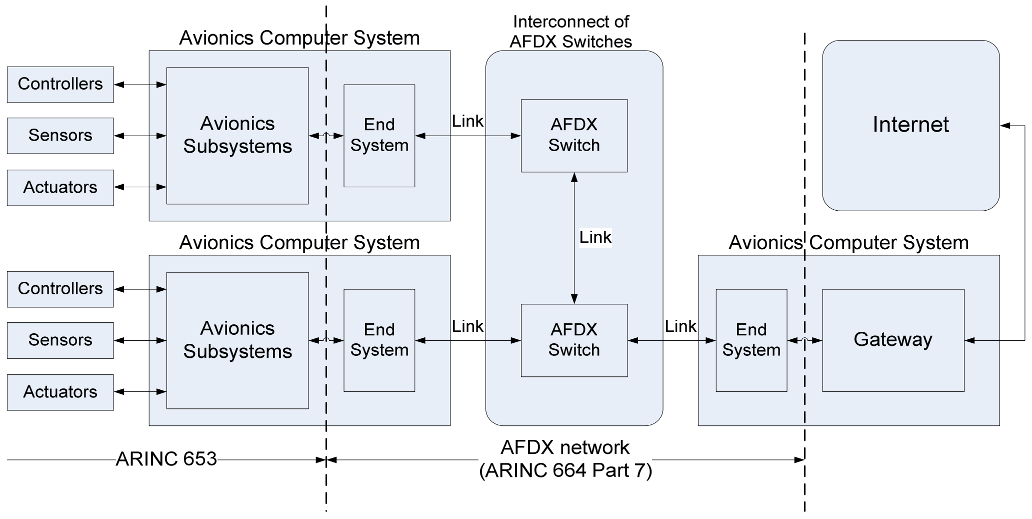


Fig. 3.29: The Avionics Subsystems connect to each other via an AFDX network

Applications on Avionic computer systems can send and receive messages through AFDX communication ports. There are two types of AFDX communication ports defined in ARINC 653: sampling ports and queuing ports. Both sampling and queuing ports are mapped to the UDP ports of the end systems in an Avionic computer system, as depicted in Fig. 3.30. The sampling and queuing ports differ mainly in message reception. Both ports are explained as follows:

- *Sampling port*: Provides a storage buffer for a single message. A new arriving message overwrites a message currently stored in the buffer. Reading the message in a sampling port does not remove the message from the buffer. Each sampling port has a freshness indicator. The freshness indicator is set when a new message is received, and it is cleared when the stored message is read. With such a freshness indicator, an End system can identify whether the transmitting Avionics subsystem has stopped transmitting, or repeatedly sent a message with the same value.

- *Queuing port*: Has queue buffers for a fixed number of messages, specified by a configuration parameter. A new arriving message is appended in the queue buffer of a queuing port. Handling all messages in a queuing port is done on First-In First-Out (FIFO) basis. Contrary to a sampling port, reading a message from a queuing port is to remove the message from the buffer.

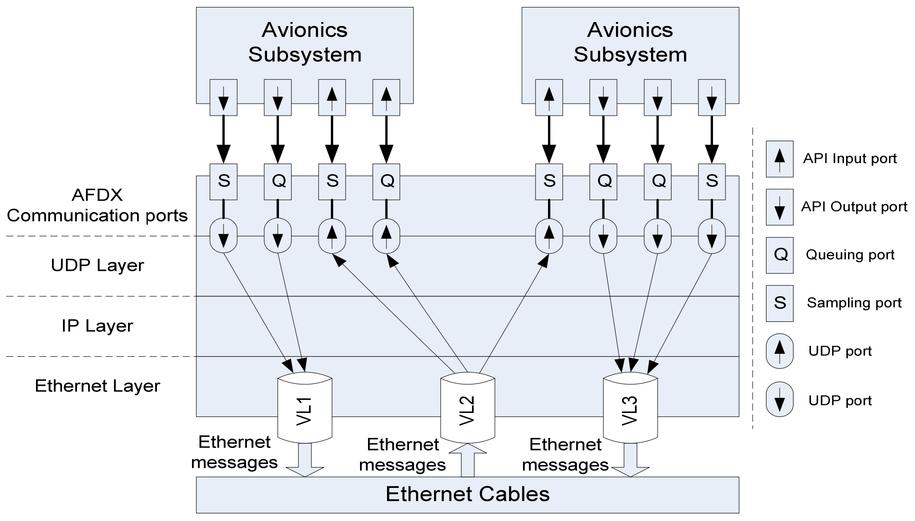


Fig. 3.30: The AFDX Communication ports at an End System

AFDX also provides another port type, called the *Service Access Point* (SAP). This is used in an End System for communicating with non-AFDX systems. An example application that uses SAP ports is TFTP (Trivial File Transfer Protocol). Connecting a legacy Ethernet network with an AFDX network can be done using a Gateway and an End System, as shown in Fig. 3.29. In a standard-Ethernet switch, an incoming Ethernet frame is routed to one or more output ports of the Ethernet switch, based on its destination MAC address. Standard-Ethernet switches use the MAC address learning mechanism, and recognize which output ports the incoming Ethernet frame should be sent to. The MAC address learning mechanism causes uncertain delays in Ethernet switches. In AFDX networks, a concept called *Virtual Link (VL)* is used for predefining routes of Ethernet frames in switches. There may be one or many VLs in an AFDX switch. A VL is a logical unidirectional connection from one source end system to one or more destination end systems, as depicted in Fig. 3.31.

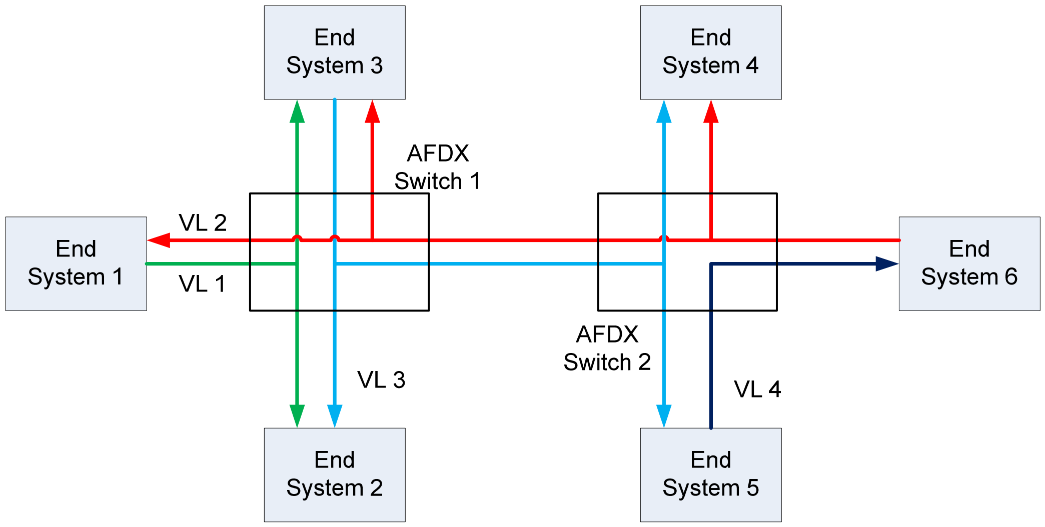
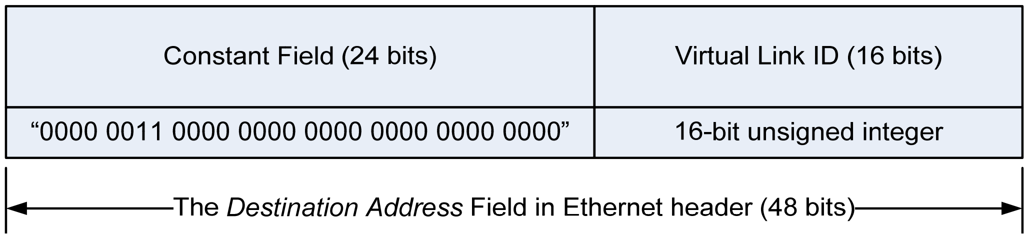


Fig. 3.31: Four Virtual Links (VL 1, VL 2, VL 3 and VL 4) in an AFDX network

Each VL is identified by a value in the *Virtual Link ID* field, which is defined as a 16-bit value in the *Destination Address* field of the Ethernet frame format (see Fig. 3.32). Looking at Fig. 3.31, the Virtual Link ID of VL 1 is 100, so messages with this Virtual Link ID from End System 1 are always relayed by AFDX switch 1 to End System 2 and End System 3.



Constant Field (24 bits)	Virtual Link ID (16 bits)
“0000 0011 0000 0000 0000 0000 0000 0000”	16-bit unsigned integer

The *Destination Address* Field in Ethernet header (48 bits)

Fig. 3.32: The *Destination Address* field in Ethernet frame format for AFDX

AFDX messages are based on Ethernet and the UDP. Avionics Subsystems send their messages to connected End Systems via AFDX ports. The End Systems encapsulate the messages with Ethernet frames before transmitting them through the AFDX network. A message from an Avionics subsystem is embedded in the *UDP Payload* field, and has a variable size of between

17 and 1471 bytes, as shown in Fig. 3.33. If a message comprises less than 17 bytes, it is padded with null bytes until it reaches the required minimum size (17 bytes). This is because the specification of standard Ethernet defines the minimum size of the *Ethernet Payload* field as 46 bytes. AFDX also defines a 1-byte field called S*equence Number* (SN). This is contained in the *UDP Payload* field and is used to indicate message sequence. The SN field is located in the last part of the *UDP Payload* field. The value of the SN field is increased by one each time an End system transmits the comprising AFDX message. The SN value is initialized to zero, and will continue to 255 until is rolls back to 1. The SN value of zero is reserved as the *End System Reset* value. Each Virtual Link has an individual SN value.

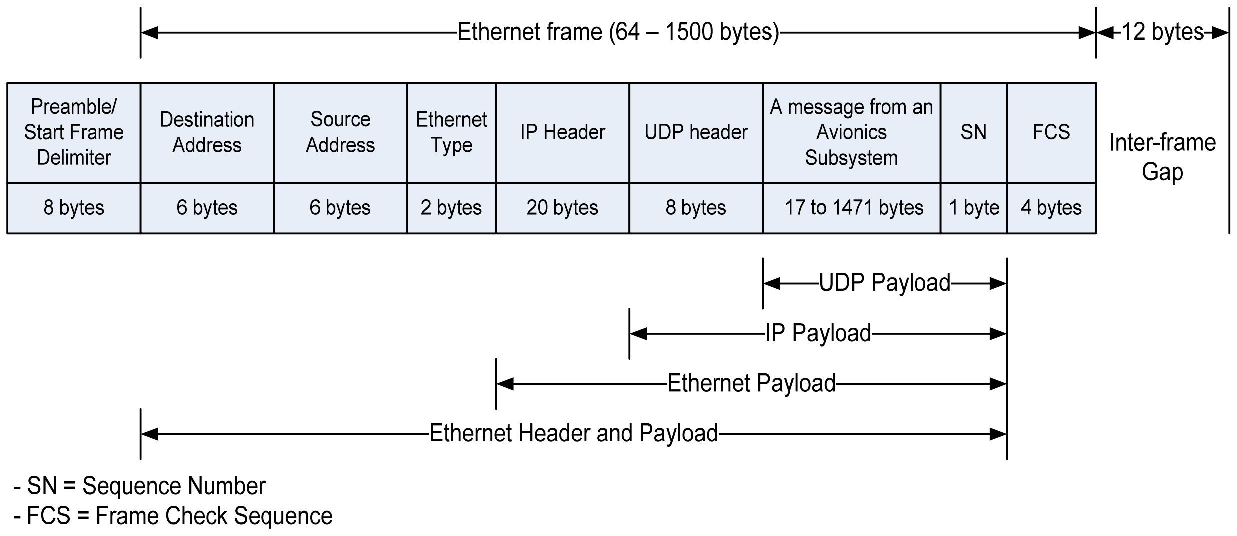


Fig. 3.33: The AFDX message format based on the Ethernet and UDP frame formats

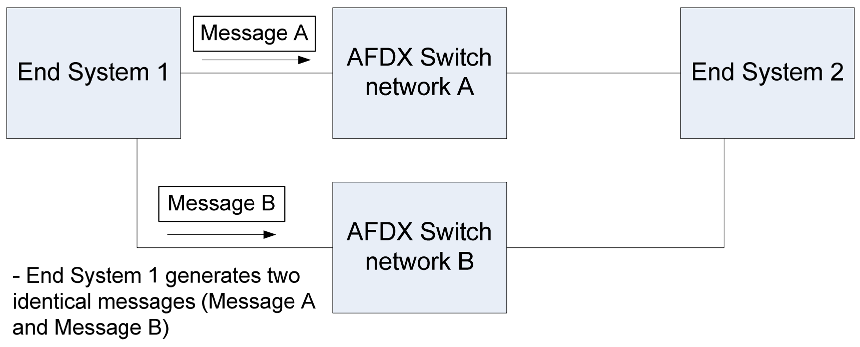


Fig. 3.34: The two redundant AFDX networks

AFDX supports a fault-tolerant property by means of message redundancy. As depicted in Fig. 3.34, two independent AFDX Switch networks (A and B) are used for transporting redundant messages in the AFDX system. Each message transmitted by End System 1 is duplicated, so there are two identical messages (Message A and Message B). Message A is sent to AFDX Switch network A, while Message B is transmitted to AFDX Switch network B. Under normal operation, End System 2 will receive the two identical messages. End System 2 then checks if the value of the *Frame Check Sequence* field of each received message corresponds with the one calculated at the MAC Layer, as shown in Fig. 3.35. Only messages with valid values are passed to the next stage, *Integrity Checking*. This stage checks whether the value in the *Sequence Number* field of a received AFDX message is in the correct, ascending order. If the received AFDX message is in the correct order, it is passed to the next stage, *Redundant Management*. Otherwise, the message is discarded. If more than one redundant message arrives at the *Redundant Management* unit, only the earliest one will be carried on to the next step. The rest will be discarded.

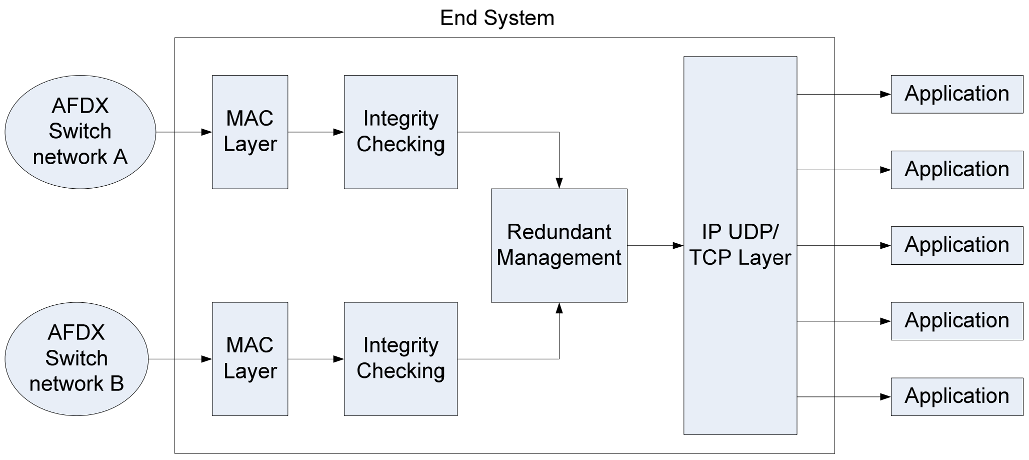


Fig. 3.35: The receiving process of the AFDX frames in an End System

Virtual Links in an AFDX network denote a logical unidirectional connection from a source End System to one or more destination End Systems. A Virtual Link is characterized by the following two parameters:

- *Bandwidth Allocation Gap (BAG):* The minimum interval between the starting bits of two consecutive Ethernet frames transmitted on a Virtual Link, as depicted in Fig. 3.37 (Traffic Regulator 1). A BAG value is equal to the power of two that results in a range from 1 to 128 milliseconds (i.e. 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6 and 2^7 milliseconds).

- *Largest Ethernet frame (Lmax):* The maximum Ethernet frame size (in bytes) that is allowed to transmit on a Virtual Link.

With these characteristics of a Virtual Link (BAG and Lmax), the maximum bandwidth utilization of a Virtual Link can be calculated using Lmax / BAG. For example, a Virtual Link assigned with a BAG of 16 milliseconds and an Lmax of 200 bytes would have a maximum bandwidth utilization of 100,000 bits per second.

Messages that are moved from Avionic subsystems to AFDX communication ports are encapsulated with UDP, IP, and Ethernet frames. These encapsulated AFDX messages will be placed in the buffers of the appropriate Virtual Links on a First-In First-Out (FIFO) basis. The transmission of AFDX messages from each Virtual Link in an AFDX End System is scheduled by the *Virtual Link Scheduler*. The Virtual Link Scheduler has to ensure that each Virtual Link in the AFDX End System conforms to its bandwidth constraints (BAG and Lmax).

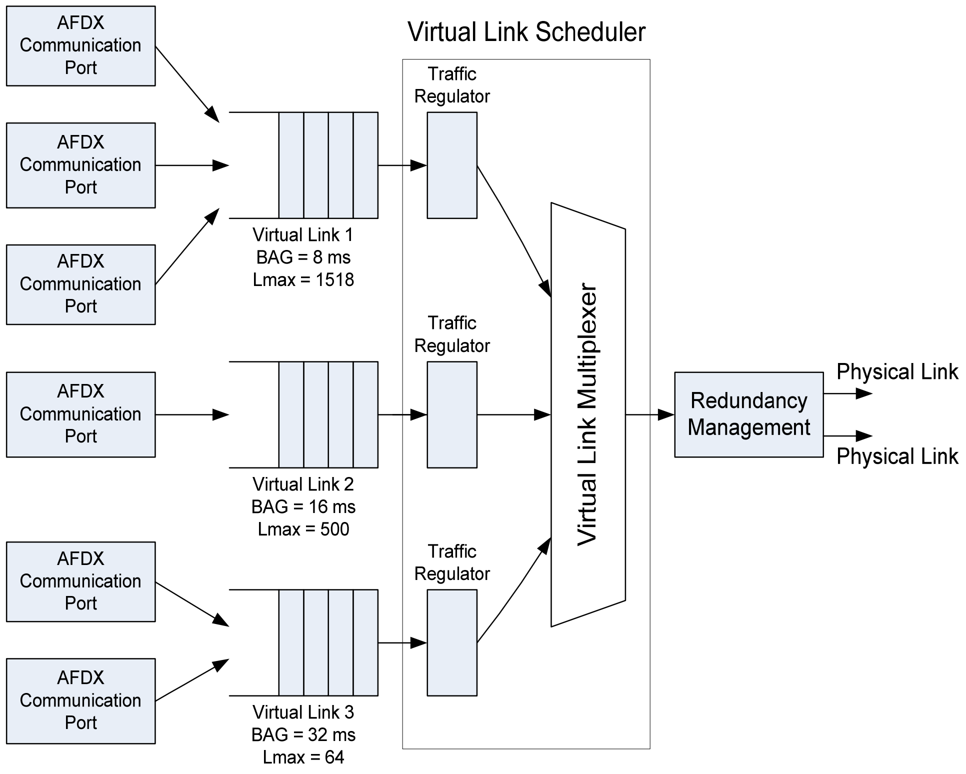


Fig. 3.36: The scheduling of the Virtual Link Schedule in an AFDX End System

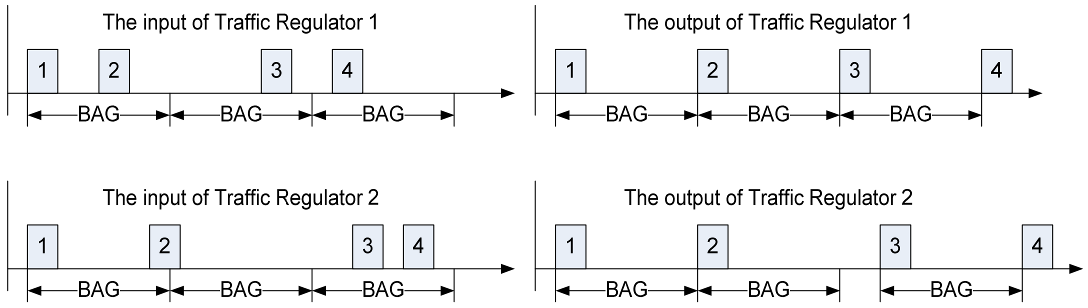

The input of Traffic Regulator 1
The output of Traffic Regulator 1
The input of Traffic Regulator 2
The output of Traffic Regulator 2
1
2
3
4
BAG

Fig. 3.37: An illustration of the inputs and outputs of two Traffic Regulators

A Virtual Link Scheduler consists of a traffic regulator for each Virtual Link, and a Virtual Link Multiplexer, as depicted in Fig. 3.36. The traffic regulator of each Virtual Link controls the time duration between any two consecutive AFDX messages, as illustrated in Fig. 3.37. The Virtual Link Multiplexer of an AFDX End System is used for multiplexing the outputs of the traffic regulators. The messages from the output of the Virtual Link Multiplexer are duplicated at the *Redundancy Management* unit, shown in Fig. 3.36. Each of the identical AFDX messages is sent to one of the independent AFDX Switch networks.

3.7.2 Performance of AFDX

With regard to real-time performance, the jitter caused by the AFDX mechanism is still an issue for analyzing a worst-case end-to-end delay [DXL10] [CXW09] [LSF09]. Jitters occurring in an AFDX End System and an AFDX switch can be examined using the following methods:

- *Jitter caused by an AFDX End System*: In an AFDX End System, Virtual Links and the Virtual Link Scheduler are the most significant causes of jitter. Jitter at a Virtual Link is introduced if a message arrives at the non-empty queue [Con05], especially if more than one sub Virtual Link is using the same Virtual Link. AFDX handles the messages from sub Virtual Links using round-robin scheduling, which is known to induce jitter [Con05]. In accordance with the ARINC 664 specification, the maximum number of sub Virtual Links in one Virtual Link is limited to four. In a Virtual Link Scheduler, jitter caused by multiplexing the multiple Virtual Links can occur when multiple Virtual Links are scheduled at the same time, meaning that their BAG values coincide. The ARINC 664 specification [ARI06] defines the maximum allowed jitter caused by multiplexing for each Virtual Link using the following formulas [Con05] [Act05]:

  $$\text{max \_jitter} \leq 40\,\mu s + \frac{\sum_{j\in\{set\ of\ VLs\}}(20+Lmax_j)*8}{Nbw} \qquad (3.1)$$

  $$\text{max \_jitter} \leq 500\,\mu s \qquad (3.2)$$

  Nbw is the link bandwidth of Ethernet (e.g. 100 Mbps for Fast Ethernet). Note that formula 3.2 denotes the highest bound of the maximum allowed jitter of each Virtual Link.

- *Jitter caused by an AFDX Switch*: Jitter in an AFDX Switch is introduced where more than one Virtual Link uses the same output port. More precisely, jitter can occur when messages from different Virtual Links are sending at the same output port of an AFDX switch simultaneously. AFDX handles this message contention using *Traffic Policing* [Act05]. With *Traffic Policing,* output ports of an AFDX Switch are configured with respective maximum transmission rates (i.e. messages that cause excessive transmission rates over the configured maximum are dropped). Thus, the jitter of messages from each Virtual Link can be bounded.

Furthermore, legacy Ethernet devices cannot directly connect to an AFDX Switch because standard-Ethernet traffic will interfere with the timely behavior of AFDX traffic (i.e there is no mechanism to handle the interference from standard-Ethernet traffic in an AFDX Switch). However, legacy Ethernet devices from non-AFDX networks can connect to an AFDX network, by using two dedicated components: a gateway and an End system, as depicted in Fig. 3.29.

3.8 TTEthernet

TTEthernet [TTE08] is a time-triggered communication protocol that expands standard Ethernet [IEE05] with services to meet time-critical, deterministic, and safety-relevant conditions. TTEthernet was developed by TTTech Computertechnik AG [TTTech] as an industrial protocol. It was further developed from the academic *TT-Ethernet* research [KAGS05] [SGAK06]. TTEthernet effectively supports mixed-criticality applications coexisting in a single network infrastructure. With the concept of time-triggered communication [KG93], TTEthernet can handle a variety of temporal requirements, including those suitable for: non real-time applications (e.g. Internet applications), soft real-time applications (e.g. multimedia applications), and hard real-time applications (e.g. fly-by-wire applications). TTEthernet is fully compliant with standard Ethernet, and transparently integrates real-time traffic with standard Ethernet traffic. It was designed to obtain timely and deterministic communication [KAGS05] for real-time traffic (time-triggered traffic) in an Ethernet-based network (i.e. there is no interference to real-time traffic from non real-time traffic). By using a common time-triggered (TT) communication schedule, TTEthernet can determine which TTEthernet device is going to transmit time-triggered messages, and *a priori* predict when the time-triggered messages will arrive at their intended destinations. TTEthernet can handle non real-time traffic *without* a degradation of timing performance to real-time traffic. This implies that the throughput of real-time traffic in a TTEthernet network is constant, because of the very low jitter. In addition, TTEthernet provides fault-tolerance approaches for safety-relevant applications, such as fault-tolerant clock synchronization, network redundancy, fault-tolerant startup, and so on [TTE08]. With regard to standard-Ethernet traffic, TTEthernet ensures high channel utilization. This is because standard Ethernet traffic can flow through TTEthernet devices irrespective of whether the system-wide global time base has been established. TTEthernet has recently been standardized as SAE 6802 [AVI11].

3.8.1 Structure and Functionality of TTEthernet

TTEthernet classifies message traffic into three classes: time-triggered messages, rate-constrained messages, and best-effort messages. In compliance with standard Ethernet, every message in a TTEthernet network is encapsulated with a standard Ethernet frame before transmitting, as depicted in Fig. 3.38. Each of the message classes are explained as follows:

- *Time-triggered messages* (TT messages): Used for transmitting real-time data. The transmissions of all TT messages are scheduled without confliction in a common TT-communication schedule. TT messages take precedence over messages of other traffic classes. Therefore, the transport delays of TT messages between two end systems are known *a-priori*, and have a very small jitter. In addition, TT messages of an end system are transmitted when the system-wide synchronized time-base at the end system has been established.

- *Rate-constrained messages* (RC messages): Used for applications which require a bounded message transmission rate (e.g. multimedia applications). Another example of applications using RC messages is AFDX [Con05], where Virtual Links for transmitting messages require a bounded transmission rate for assigning the *Bandwidth Allocation Gap*. Applications using RC messages have less stringent timing-determinism and real-time requirements than ones using TT messages.

- *Best-effort messages* (BE messages): Used for standard Ethernet traffic. BE messages are transmitted without timing or transmission-rate constraints. There is no guarantee of arrival times for BE messages at their specified end systems. Therefore, BE messages are only suited for non real-time traffic. In a TTEthernet network, BE messages have less priority than TT and RC messages, and use the bandwidth remaining after their transmission.

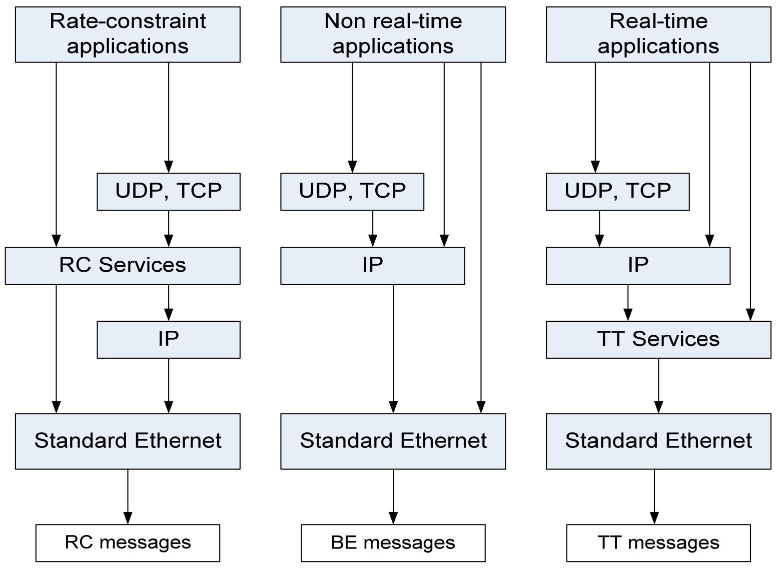


Fig. 3.38: The protocol layers of TTEthernet compliant with standard Ethernet

Fig. 3.39 shows a TTEthernet system consisting of a set of end systems and TTEthernet switches (TTE switches). The end systems connect with each other via the TTE switches. The network structure of TTEthernet, which complies with switched Ethernet, is in a star topology. A TTEthernet end system consists of a host computer and a TTEthernet controller. The host computer of a TTEthernet end system executes applications, while the communication controller is responsible for executing the TTEthernet communication protocol. The TTEthernet controller contains all time-triggered communication services. The applications in the host computer run independently of the TTEthernet controller. The TTEthernet controller and the host computer are connected via a communication network interface (CNI). There is full-duplex message communication between each end system and its connected TTE switches. All TTEthernet end systems and TTE switches handle TT traffic in consideration with the common TT communication schedule. TT messages can be transmitted at a TTEthernet End System, or relayed by a TTE switch if and only if the system-wide global-time base has been established by the clock synchronization mechanism [TTE08].

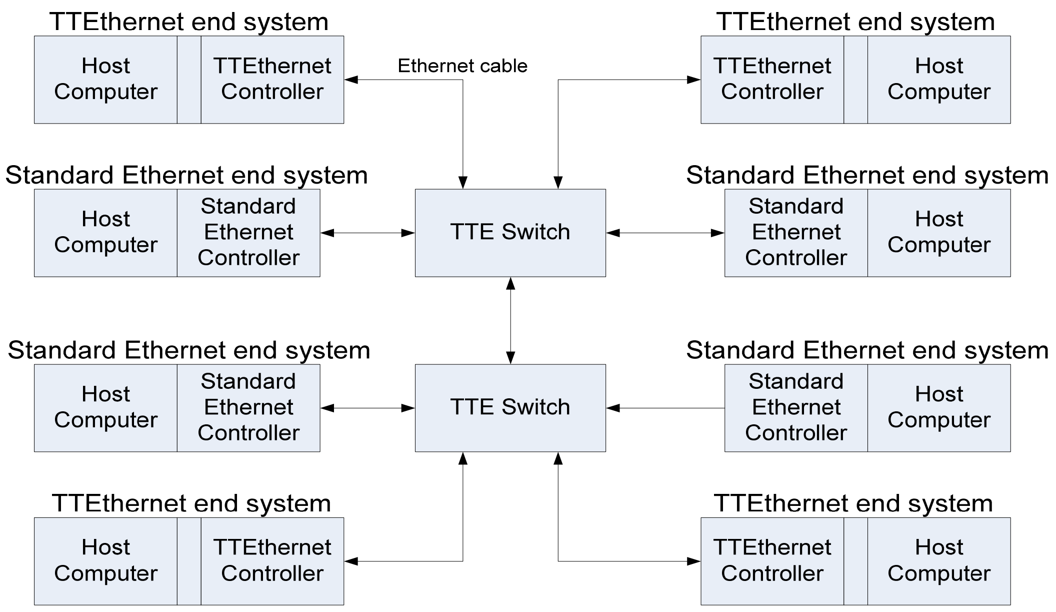


Fig. 3.39: Legacy Ethernet devices coexisting with TTEthernet end systems in a TTEthernet system

TTEthernet also supports two redundant communication channels, as shown in Fig. 3.40. Therefore, faults occurring in one of two redundant communication channels (e.g. a break in a network cable) can be tolerated.

The clock synchronization of TTEthernet can be configured to operate as master-slave clock synchronization or multi-master clock synchronization. Therefore, the clock synchronization of

TTEthernet can be used in a broad range of applications. Master-slave clock synchronization, which is simple to configure, is mostly used in industrial control applications. Multi-master clock synchronization is suitable for highly fault-tolerant systems (e.g. fly-by-wire systems). The system-wide global-time base in a TTEthernet network is performed on the *cluster cycle* specified in the common TT communication schedule. The definition of the cluster cycle is explained in Chapter 2 – *Basic Concepts*. Both synchronization and TT messages are transmitted and received with regard to their assigned instants along the cluster cycle of a real-time cluster, as illustrated in Fig. 3.41. A real-time cluster in a TTEthernet system is called a *TTEthernet cluster*. TTEthernet defines a dedicated synchronization message, called the *Protocol Control Frame* (PCF). A PCF contains accumulated time information regarding travel from sender to receiver, as depicted in Fig. 3.42. The PCF field containing this accumulated information is called the *transparent clock*. A TTEthernet end system or switch uses the PCF's *transparent clock* of a received message to calculate its time correction value. In additional, TTEthernet applies the concept of *Message Permanence* (explained in Chapter 2 – *Basic Concepts*). This is used to reorder messages received at a TTEthernet end system or TTE switch. This denotes that messages arriving at a TTEthernet end system can be rearranged in transmission order (as observed by an omniscient external observer [Kop97]). The function of TTEthernet that determines the permanence points in time for receiving messages is called the *Message Permanence Function*. This means that a receiving message becomes permanent at its permanence points in time, when the receiver has already received all prior messages to the current one.

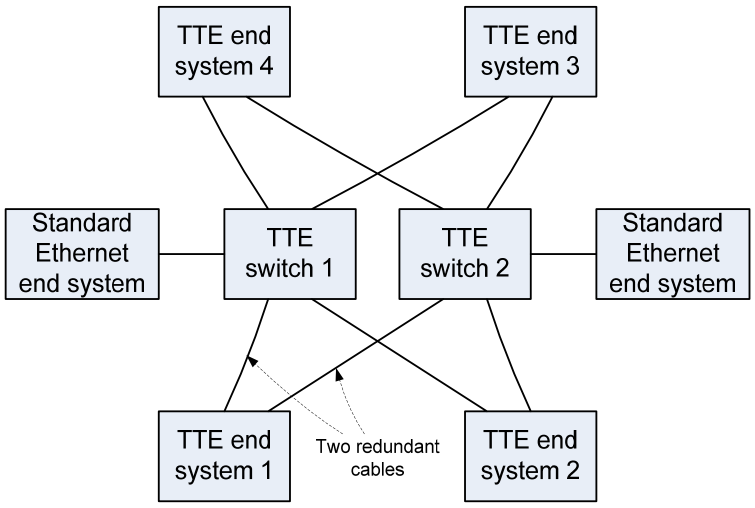


Fig. 3.40: The two communication channels in a TTEthernet system

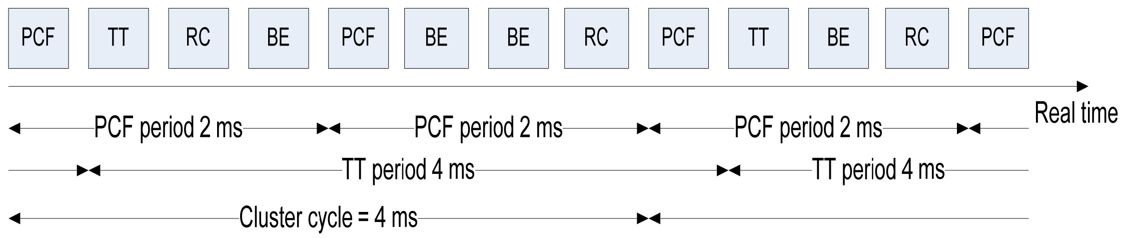


Fig. 3.41: The cluster cycle in a TTEthernet system

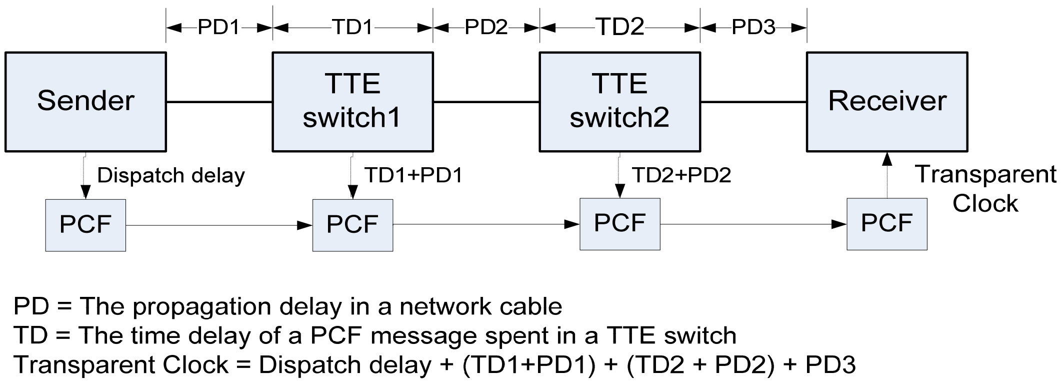

PD1
TD1
PD2
TD2
PD3
Sender
TTE switch1
TTE switch2
Receiver
Dispatch delay
TD1+PD1
TD2+PD2
Transparent Clock
PCF
PCF
PCF
PCF
PD = The propagation delay in a network cable
TD = The time delay of a PCF message spent in a TTE switch
Transparent Clock = Dispatch delay + (TD1+PD1) + (TD2 + PD2) + PD3

Fig. 3.42: The Transparent Clock of a PCF in a TTEthernet system

The most important parameter in the *Message Permanence Function* is the *maximum transmission delay*. The maximum transmission delay is a bounded transmission delay. It is either equal to or greater than the maximum value of the transport delays of PCFs travelling from all senders to all receivers in a TTEthernet cluster. In Fig. 3.42, the receiver uses the *transparent clock* value of a receiving PCF to calculate its permanence point in time, as detailed by Equation 2.2 (located in Chapter 2 - *Basic Concepts*).

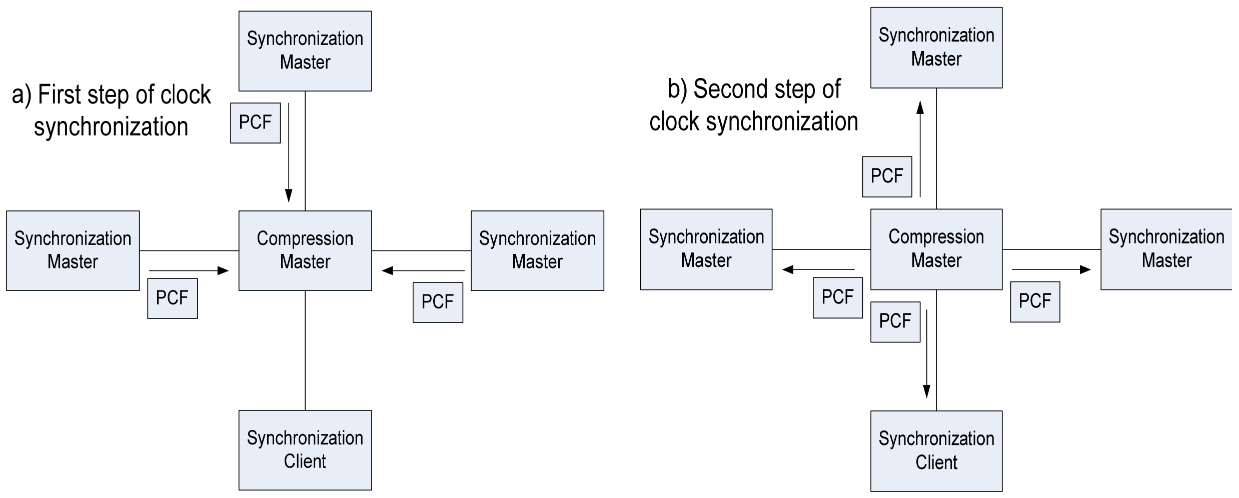


Fig. 3.43: Two steps of the clock synchronization in a TTEthernet system

There are three types of TTEthernet clock synchronization devices: synchronization master, compression master, and synchronization client. Fig. 3.43 depicts two steps of the clock synchronization process in a TTEthernet system. In the first step, the synchronization masters send their respective PCFs to the compression master, according to the specified sending instants in the common TT communication schedule. The compression master then calculates a fault-tolerant averaging value [Kop97] from the relative arrival times of the incoming PCFs (i.e. from their relative *permanence points in time*). The compression master then corrects its clock time. In the second step, the compression master sends the new PCFs back to the synchronization masters, as well as to the synchronization clients. The synchronization masters and synchronization clients receive the PCFs from the compression master, and then perform the corrections to their local clock times.

In a TTEthernet system, the TTEthernet devices perform their startup services as soon as they are powered. The TTEthernet startup service is responsible for establishing the system-wide global-time base for all TTEthernet devices. TTEthernet provides three PCF frame types for this startup service: cold-start frame, cold-start acknowledgement frame, and integration frame. Only synchronization masters perform the initial transmission of these frames. In the startup state of TTEthernet, synchronization masters use cold-start frames and cold-start acknowledgement frames to obtain the global consistent point in real time for initial clock synchronization. TTEthernet provides the *fault-tolerant handshake* fault tolerance mechanism for the startup service. The details of the TTEthernet startup service can be found in the TTEthernet specification [TTE08].

TTEthernet provides a clique detection service for detecting disjoint subsets of TTEthernet devices that are synchronized within their respective subsets in a TTEthernet cluster (synchronization domain). A TTEthernet device performs this clique detection service by checking the membership state specified in a receiving PCF with its current membership state. TTEthernet also provides a clique resolution service, which re-establishes the clock synchronization in a synchronization domain after cliques have been detected. Therefore, there will be only one set of synchronized TTEthernet devices in a synchronization domain. TTEthernet defines the clique detection and resolution services as part of the startup service. Both clique detection and resolution services enhance the reliability of the system-wide global-time base in a TTEthernet system.

3.8.2 Performance of TTEthernet

Based on the time-triggered communication paradigm [Kop08] [KG93], the timing performance of TT traffic in a TTEthernet network does not depend on the amount of standard Ethernet traffic (i.e. standard-Ethernet frames cannot interfere with the timely behavior of time-triggered messages). TTEthernet devices send time-triggered messages with regard to a common confliction-free time-triggered (TT) communication schedule. Each TTEthernet device knows when time-triggered messages are transmitted, and *a priori* predict the arrival time of each time-triggered message. Therefore, the timing performance of time-triggered traffic in TTEthernet is maximized, and has very low jitter. With the determinism property of time-triggered communication [KAGS05] in TTEthernet, it is suitable for fault-tolerant real-time systems (because it supports network redundancy). With regard to the performance of standard-Ethernet traffic in a TTEthernet network, although it has lower priority than time-triggered traffic, TTEthernet still achieves high channel utilization. This is because the standard-Ethernet traffic can flow through TTEthernet devices, regardless of whether or not their system-wide global-time base has been established. Currently, there are both hardware and software-based versions of TTEthernet devices available, which are provided by the TTTech Company [TTTech]. According to [BSK11], the end-to-end latencies of TT traffic in a 100-Mbps TTEthernet network with a software-based TTEthernet end system and a hardware-based TTE switch remain approximately constant, with a jitter of less than 10 microseconds. This jitter can be reduced to less than a few microseconds by using a hardware-based TTEthernet end system instead of a software-based one. In addition, both 1-Gbps TTE switches and 1-Gbps TTEthernet end systems are currently available, and suitable for very high-speed mixed-criticality applications [TTTech].

CHAPTER 4

A Simulation Model for TTEthernet Systems

In this chapter we propose a simulation model for TTEthernet systems. Our simulation model, which is based on the concept of discrete event simulation, is built to describe the behavior of a real world TTEthernet system. This chapter begins with an explanation of the relationship between simulation time and local-clock time. We then present the simulation model of the TTEthernet system components. We use the Java library "J-Sim" (as described in Chapter 2 - Basic Concepts) and a Java runtime environment for implementing our simulation model. We calibrate this model using existing results from published papers [LL08][Rug08].

4.1 The relationship between simulation time and local-clock time

The *simulation time* of our simulation model represents the progression of real time, which is measured by a reference clock (a clock with a drift rate of zero). In a TTEthernet (TTE) system, each TTE device (either a TTE end system or a TTE switch) has its own local clock, which expresses its *local-clock time*. Each local clock in every TTE device has a varying drift rate. The varying drift rate of a local clock is bounded by its specified maximum drift rate. In our simulation model, the local clock of a TTE device performs the local-clock time progression using its specified maximum drift rate. Therefore, the local-clock time will deviate from a reference clock within the drift offset value derived from its specified maximum drift rate, at a particular time interval of interest. We depict the relationship of simulation time and local clock time in Fig. 4.1. We obtain the local-clock time of a local clock using Equation 4.1.

$T_{clock\_current} = T_{clock\_last} + (T_{sim\_current} - T_{sim\_last}) * (1+ \rho)$ ....(4.1)

Where,

$T_{clock\_current}$ is the local-clock time at the current simulation time ($T_{sim\_current}$)

T_{clock_last} is the local-clock time at the last simulation time (T_{sim_last})

$T_{sim\_current}$ is the current simulation time

T_{sim_last} is the last simulation time at the last update of a local-clock time.

ρ = the specified maximum drift rate of a local clock

Due to clock synchronization, the last local-clock time (T_{clock_last}) is updated by the corrected local-clock time ($T_{clock\_current}$) at a resynchronization point in simulation time ($T_{sim\_current}$). The last simulation time (T_{sim_last}) is then updated with the current simulation time ($T_{sim\_current}$). The details of how to obtain the corrected local-clock time are described in the *local-clock component* section.

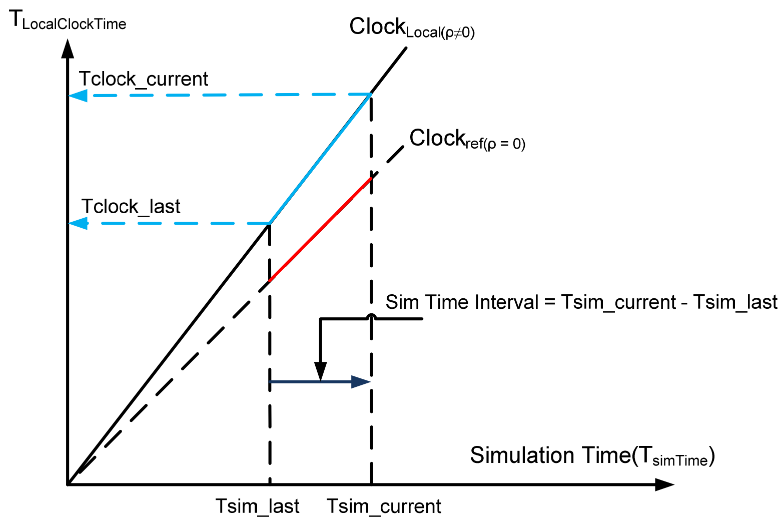


Fig. 4.1: The relationship between simulation time and local clock time

4.2 The simulation model of TTE system components

In this section we describe our simulation model of all TTE system components. Firstly, the simulation model structure, which is based on *discrete event simulation,* is presented. The simulation model structure for a TTE system mainly consists of a cluster list, a network-cable list, and two main methods (i.e. *Life* and *Init*). These are depicted in Fig. 4.2. In a discrete event simulation, there is an event list containing all events to be executed. In J-Sim, this event list is called a *simulation calendar*. An event in a simulation calendar consists of a component's reference and the event's point in simulation time. When the event's point in simulation time is reached, the *Life* method of the component indexed by the event is executed. We call the arrival of an event's point in simulation time *the event occurrence*.

- Cluster List: We model a TTE cluster as a *cluster* component. A *cluster* component is mainly composed of a set of *node* and *TTE-switch* components. A cluster list is a container of all cluster components in a TTE system's simulation. Events that are requested from any component in a cluster list are added into the simulation calendar (as depicted in Fig. 4.2). These events occur at their event's points in simulation time, i.e. the components of these events are activated to be executed.

- Network-cable List: We model an Ethernet network cable as a *network-cable* component. The network-cable list in a TTE system's simulation contains all *network-cable* components. Similar to the cluster list, events that are requested from a *network-cable* component are added into the simulation calendar (as depicted in Fig. 4.2). The components of these events are activated to be executed at their event's points in simulation time.

- Event List: All event processes are held in an event list, and are executed with respect to their event's point in simulation time. We use the J-Sim simulation calendar as an event list.

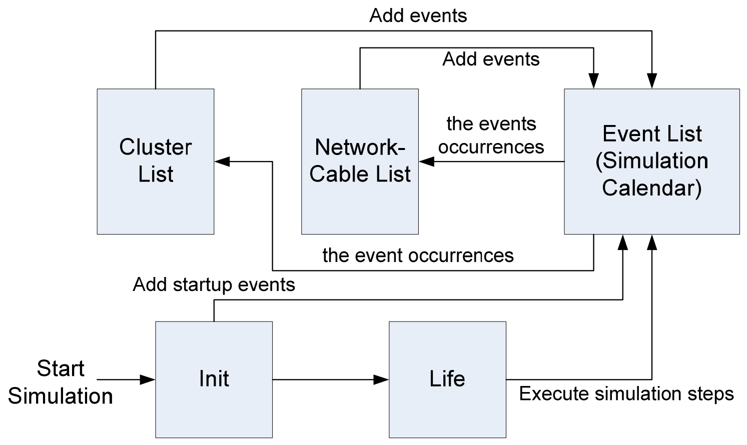


Fig. 4.2: The simulation model structure for a TTEthernet system

- *Life* method: The *Life* method is the main method of a TTE system's simulation. This method executes simulation steps, i.e. the *Life* methods of the components are executed in ascending time order.

- *Init* method: The *Init* method is executed when a TTE system's simulation begins. This method adds the startup events of all components to the simulation calendar, in order to start their operations at user-defined startup times.

4.2.1 Cluster component

A *cluster* component consists of a set of *node* components and *TTE-switch* components, all of which connect with each other via *network-cable* components. A *cluster* component provides a node list, a TTE-switch list (for containing *node* components), and *TTE-switch* components, respectively. This is shown in Fig. 4.3. The *node* and *TTE-switch* components from different *cluster* components can also connect with each other via a *network-cable* component.

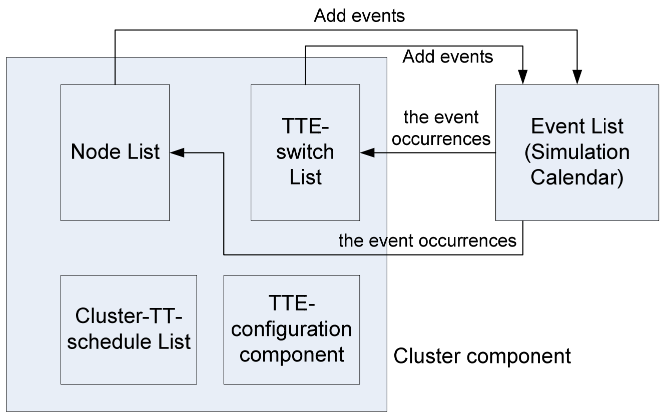


Fig. 4.3: A *cluster* component

- *Node* list: A *Node* list contains all *node* components that belong to a particular *cluster* component. Events that are requested from the *node* components are added into the simulation calendar. The components of these events are activated to be executed at their event's points in simulation time.

- *TTE-switch* list: A *TTE-switch* list contains all *TTE-switch* components that belong to a particular *cluster* component. Similar to a *node* list, events that are requested from the *TTE-switch* components are added into the simulation calendar. The components of these events are activated to be executed at their event's points in simulation time.

- *TTE-configuration* component: A *TTE-configuration* component contains the TTE configuration parameters, e.g. synchronization device type, synchronization priority, maximum transmission delay, local-clock precision, acceptance widow, integration cycle duration, and so on. These parameters are described in the *TTE-configuration component* section. Once a *TTE-engine* component starts running, it loads all TTE-configuration parameter values from the *TTE-configuration* component.

- *Cluster-TT-schedule* list: A *Cluster-TT-schedule list* belongs to a *cluster* component, and stores TT-schedule components. Each TT-schedule component contains a user-assigned TT-traffic load. The TT-schedule component is described in the *TT-schedule component* section.

4.2.2 Node component

We model a TTE end system as a *node* component. There are two main operations in a *node* component: standard-Ethernet communication, and TT communication.

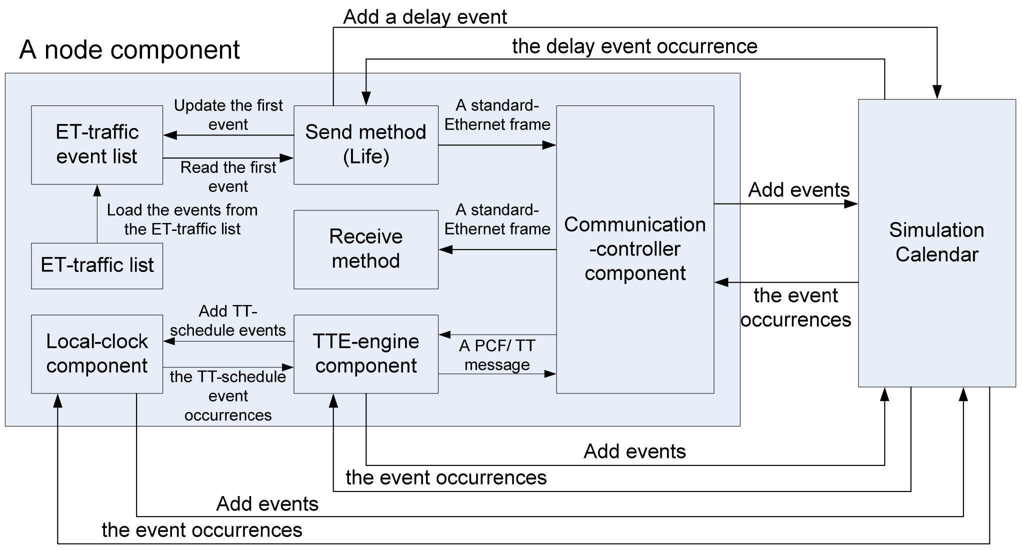


Fig. 4.4: A *node* component

Standard-Ethernet communication: The transmission of standard-Ethernet frames is based on event-triggered communication. As shown in Fig. 4.4, a *node* component has an *ET-traffic* list that stores ET-traffic components. An ET-traffic component contains a user-assigned transmission pattern of standard-Ethernet traffic. An ET-traffic component has the following parameters:

- Data Size: The payload data size of a standard-Ethernet frame.
- Transmission Rate: The transmission rate of a standard-Ethernet frame.
- Transmission Type: The transmission type of a standard-Ethernet frame: unicast, multicast, or broadcast.
- Destination Node: This parameter is specified only if the Transmission Type parameter is unicast. It indicates the destination node of a standard-Ethernet frame. We model a standard-Ethernet frame as an *Ethernet-frame* component. A *node* component can receive an incoming *Ethernet-frame* component if the *destination node* parameter of the incoming *Ethernet-frame* component matches to the *node* component. Otherwise, the node component drops the incoming *Ethernet-frame* component.
- Traffic Type: This parameter indicates the traffic type of an *Ethernet-frame* component. There are two traffic types for standard-Ethernet traffic: RC (Rate-constrained) and BE (Best-effort). The RC traffic type is for high-priority standard-Ethernet traffic, whereas the BE traffic type is for low-priority standard-Ethernet traffic.

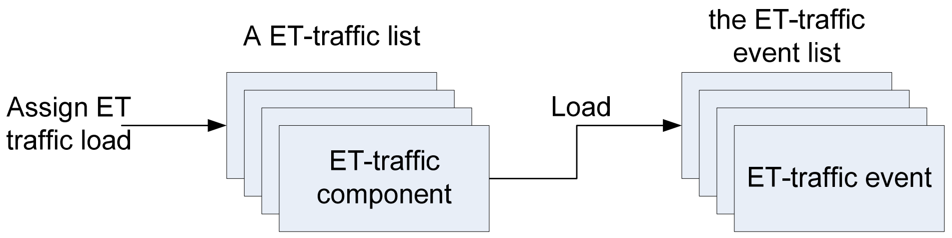


Fig. 4.5: The derivation of the ET-traffic events in the ET-traffic event list

As shown in Fig. 4.5, once a *node* component begins its operation, it initializes ET-traffic events in the ET-traffic event list. An ET-traffic event consists of an ET-traffic component's reference, and an ET-traffic event's point in simulation time. At the ET-traffic event's point in simulation time, the *node* component dispatches an *Ethernet-frame* component, as depicted in Fig. 4.5. In event-triggered communication, we define that the ET-traffic event's point in time is the same as the ET-traffic event's point in simulation time. All ET-traffic events in the ET-traffic event list are in ascending order with respect to their event's point in time. A node component considers the ET-traffic event with the lowest point in time, and dispatches an Ethernet-frame component. After every node component dispatches an Ethernet-frame component, the ET-traffic event with the lowest point in time is updated by a time delay, as depicted in Fig. 4.6.

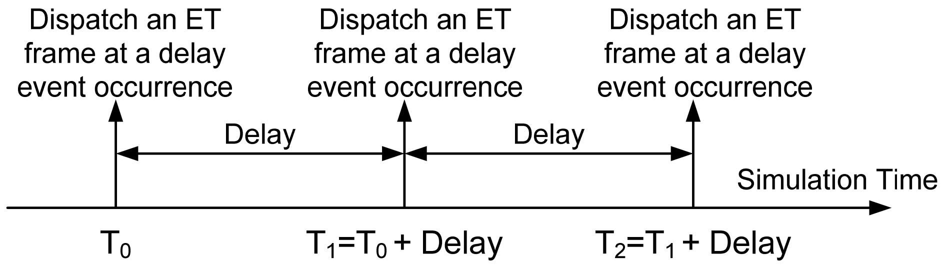

Dispatch an ET frame at a delay event occurrence
Dispatch an ET frame at a delay event occurrence
Dispatch an ET frame at a delay event occurrence
Delay
Delay
Simulation Time
T_0
T_1=T_0 + Delay
T_2=T_1 + Delay

Fig. 4.6: Timing diagram of transmitting Ethernet frames

In a *node* component, there are two methods for generating and receiving standard-Ethernet messages, these are the *Send* method and *Receive* method, respectively.

- *Send* method: This method generates and dispatches a standard-Ethernet frame to the *communication-controller* component, as depicted in Fig. 4.4. Firstly, the ET-traffic components in an ET-traffic list are read. Then a time delay between the dispatching points in time of two consecutive standard-Ethernet frames is calculated. Finally, *Ethernet-frame* components are dispatched with a period of the calculated time delay, as depicted in Fig. 4.6. The time delay is derived from the *Transmission Rate* parameter of an *ET-traffic* component. We model a standard-Ethernet frame as an Ethernet-frame component, as described in the *Ethernet-frame component* section.

- *Receive* method: This method performs the standard-Ethernet reception service of a *node* component, when the *communication-controller* component receives an entire standard-Ethernet frame (as depicted in Fig. 4.4). This standard-Ethernet reception service includes a process to record receiver information for analysis purposes.

TT communication: In a node component, the *TTE-engine* component dispatches and receives TT messages and PCFs to/from the *communication-controller* component (as depicted in Fig. 4.4). The TT communication of a *node* component is described in the following sections: TTE-engine component, communication-controller component, and local-clock component.

The parameters of a *node* component: In our simulation model we provide the following parameters for a *node* component:

- Drift Rate: The local-clock drift rate of a *node* component. The parameter unit takes the form of second/second (sec/sec).
- Startup Time: Determines a startup point of node operation in simulation time. If simulation time has not reached the startup time of a *node* component, all operations of the underlying components (i.e. local-clock, communication controller, and TTE-engine components) cannot get started.
- Communication Protocol: Determines a communication protocol used in a *communication-controller* component. There are two communication protocols: Ethernet (for standard-Ethernet devices), and TTEthernet (for TTE devices). If this parameter is set to Ethernet, the *communication-controller* component performs as a standard-Ethernet controller. If the parameter is set to TTEthernet, the *communication-controller* component performs as a TTEthernet controller. In a TTEthernet controller, the time-triggered traffic can coexist with standard-Ethernet traffic.

4.2.3 Network-Cable component

We model an Ethernet network cable as a *network-cable* component. A *network-cable* component consists of two *transmission-line* components and two *connector* components, as depicted in Fig. 4.7. The *network-cable* component supports full-duplex communication. One of two underlying *transmission-line* components takes a message from one end to another (in one direction), while another *transmission-line* component delivers a message in the opposite direction. A *connector* component has a connection parameter that identifies which *communication-controller* component or *TTE-switch-port* component it connects to.

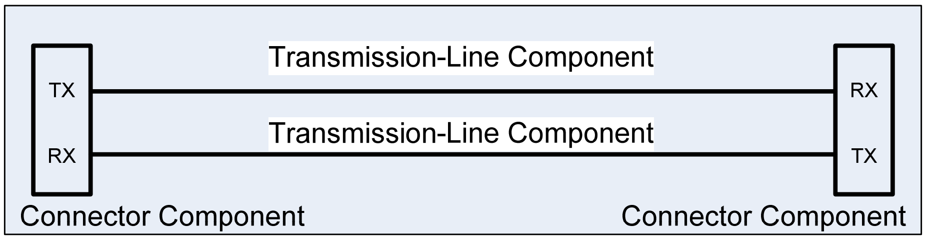


Fig. 4.7: The internal structure of a *network-cable* component

Transmission-Line component: A *transmission-line* component relays a message from the connected *transmitter* component to the connected *receiver* component. The relaying time is specified in the *propagation delay parameter* of the *transmission-line* component. A *transmission-line* component has one event list and two main methods (i.e. *message-propagation service*, and *Life*), as depicted in Fig. 4.8. The *message-propagation service* method is executed when a connected *transmitter* component transmits a message component. Note that a message component is an *Ethernet-frame*, a *PCF*, or a *TT-message* component. The *message-propagation service* method of a *transmission-line* component is invoked by the connected *transmitter* component. This method adds a transmission-line event into the simulation calendar, and the message component into the message list. The transmission-line event consists of the *transmission-line* component's reference and the receiving point in simulation time. The receiving point in simulation time is equal to the point in simulation time when a message arrives at the *transmission-line* component plus the parameter value of propagation delay. We call the execution of the *Life* method at this receiving point in simulation time the *transmission-line event occurrence*. When a *transmission-line event* occurs, the *Life* method of the *transmission-line* component forwards the corresponding message component to the connected receiver component. Note that a *network-cable* component has two *propagation delay* parameters, one for each *transmission-line* component.

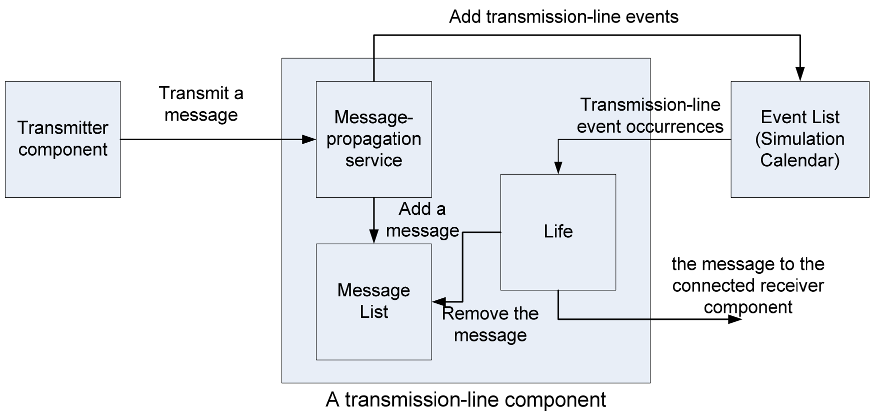

Add transmission-line events
Transmitter component
Transmit a message
Message-propagation service
Transmission-line event occurrences
Event List (Simulation Calendar)
Add a message
Life
Message List
Remove the message
the message to the connected receiver component
A transmission-line component

Fig. 4.8: A *transmission-line* component

Connector component: A *connector* component has two main parameters: *receiver-connector* (RX) and *transmitter-connector* (TX). The *receiver-connector* parameter indicates which *transmission-line* component of a *network-cable* component connects to the *receiver* component of a *node* or *TTE-switch-port* component. The *transmitter-connector* parameter indicates which *transmission-line* component of a *network-cable* component connects to the *transmitter* component of a *node* or *TTE-switch-port component*.

4.2.4 Local-Clock component

We model the local clock of a node or TTE switch as a *local-clock* component. A *node* or *TTE-switch* component has its own individual *local-clock* component. The specified maximum drift rate of a *local-clock* component is a parameter used for generating the local-clock time. The *local-clock* component handles a time-triggered (TT) schedule event that requests from the *TTE-engine* component (both components are in the same TTE device). A TT schedule event consists of a TT communication service and a TT schedule event's point in time. A *local-clock* component creates a local-clock event when it obtains a TT schedule event. A local-clock event consists of the *local-clock* component's reference and the TT schedule event's point in simulation time. The *local-clock* component inserts the TT schedule event into the TT schedule event list, and adds the local-clock event into the simulation calendar. When the local-clock event in the simulation calendar occurs, the *Life* method of the *local-clock* component removes the corresponding TT schedule event. The *Life* method then triggers execution of the TT communication service in the *TTE-engine* component. We provide the following methods in a local-clock component:

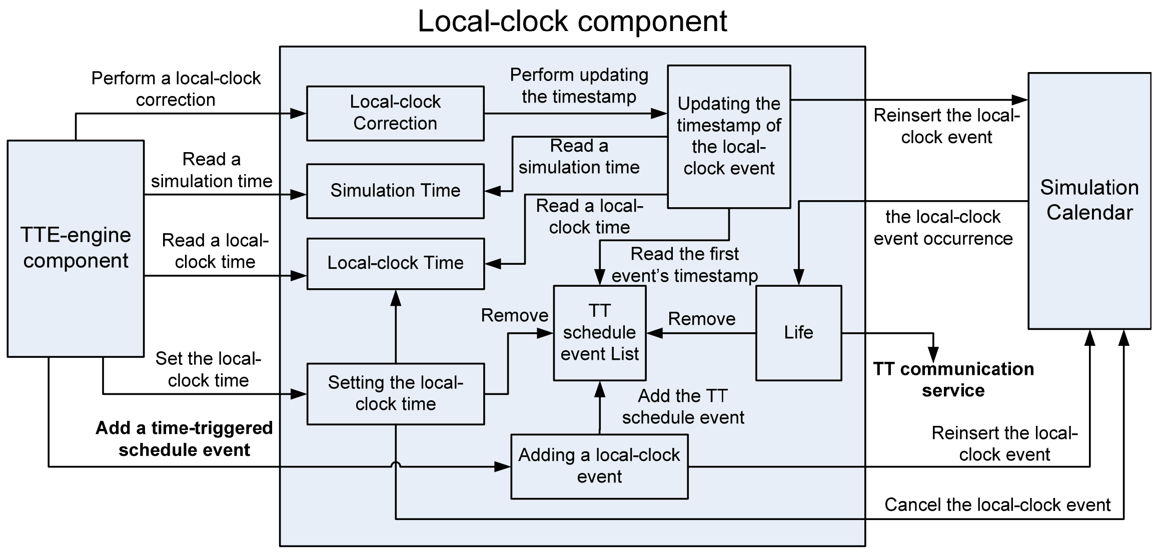

Local-clock component
Perform a local-clock correction
Read a simulation time
Read a local-clock time
Set the local-clock time
Add a time-triggered schedule event
TTE-engine component
Local-clock Correction
Simulation Time
Local-clock Time
Setting the local-clock time
Perform updating the timestamp
Read a simulation time
Read a local-clock time
Read the first event's timestamp
Updating the timestamp of the local-clock event
Remove
TT schedule event List
Remove
Life
Add the TT schedule event
Adding a local-clock event
Reinsert the local-clock event
the local-clock event occurrence
TT communication service
Reinsert the local-clock event
Cancel the local-clock event
Simulation Calendar

Fig. 4.9: A *local-clock* component

- *Local-clock correction*: Correct the local-clock time of a *local-clock* component so that it is synchronized with all others in a TTE cluster. The clock correction term (clock_correction) is a result of the clock synchronization from a *TTE-engine* component. The clock correction term is used to update the local-clock time ($T_{clock\_current}$). The updated local-clock time (T_{clock_update}) is calculated using Equation 4.2. The local-clock time ($T_{clock\_current}$) is derived from Equation 4.1

$$T_{clock\_update} = T_{clock\_current} + clock\_correction \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(4.2)$$

If the clock correction term (clock_correction) is a positive value, it denotes that the *local-clock* component runs slower than the majority of all *local-clock* components, and vice versa. After obtaining the updated local-clock time (T_{clock_update}), two variables (T_{clock_last} and T_{sim_last}) referred to in Equation 4.1 are replaced with T_{clock_update}, and $T_{sim\_current}$, respectively.

- *Updating the local-clock event in the simulation calendar*: Due to the effect of local-clock correction, the local-clock event's point in simulation time has to be updated. This is done by canceling the local-clock event in the simulation calendar. The local-clock event's point in simulation time is recalculated using Equation 4.3. The local-clock event with the new point in simulation time is then reinserted into the simulation calendar.

$$T_{sim\_time} = T_{sim\_current} + \frac{(T_{local\_clock\_time} - T_{local\_clock\_current})}{(1+dirft\_rate)} \ldots\ldots\ldots\ldots\ldots(4.3)$$

, where T_{sim_time} is an updated event's point in simulation time

$T_{sim\_current}$ is a current simulation time

$T_{local\_clock\_time}$ is a scheduled event's point in local-clock time

$T_{local\_clock\_current}$ is a current local-clock time

drift_rate is the *drift rate* parameter of a *local-clock* component

- *Adding a local-clock event:* A *local-clock* component generates a local-clock event that corresponds to a time-triggered schedule event received from the *TTE-engine* component. A time-triggered schedule event consists of a TT communication service and a TT schedule event's point in simulation time. A local-clock event consists of the *local-clock* component's reference and the TT schedule event's point in simulation time. The *local-clock* component inserts the generated local-clock event into the local-clock event list. All local-clock events in the local-clock event list are sorted in ascending order with regard to their points in simulation time. This method cancels the local-clock event in the simulation calendar. The local-clock event with the lowest point in simulation time is then reinserted into the simulation calendar.

- *Life*: This method processes a local-clock event at its occurring point in simulation time. Initially the first TT schedule event in the TT schedule event list is removed. We call a local-clock event occurring at its point in simulation time *the TT schedule event occurrence*. When a TT schedule event occurs, this process calls its TT communication service in the referred *TTE-engine* component to perform the TT communication service.

- *Setting the local-clock time*: A *TTE-engine* component sets an initial value for the local-clock time when the *TTE-engine* component begins the synchronization process. The initial value of the local-clock time is either a scheduled dispatch point in time, or a scheduled reception point in time. After the local-clock time is set, the TT schedule event list of the *local-clock* component is cleared, and the local-clock event in the simulation calendar is canceled.

- *Simulation time*: A local-clock time can be converted to simulation time using Equation 4.1.

- *Local-clock time*: A simulation time can be converted to local-clock time using Equation 4.1.

4.2.5 Communication-Controller component

We model a communication controller as a *communication-controller* component. A communication-controller component is composed of two components: a *transmitter* component and a *receiver* component, as depicted in Fig. 4.10. The *communication-controller* component has a queue buffer for standard-Ethernet frames, as well as a buffer for PCF and TT messages.

- A queue buffer for standard Ethernet frames (*ET queue buffer*): This FIFO queue buffer is used for storing *Ethernet-frame* components, and is divided into two sub queue buffers: RC queue buffer and BE queue buffer. An *Ethernet-frame* component has a *Traffic Type* parameter, identifying which sub queue buffer should be used to store it. Removing an *Ethernet-frame* component from the ET queue buffer is to remove the first *Ethernet-frame* component from the RC queue buffer. However, if the RC queue buffer is empty, then the first *Ethernet-frame* component in the BE queue buffer is read.

- A buffer for PCF and TT messages (*TT buffer*): This buffer stores either a *PCF* or a *TT* component.

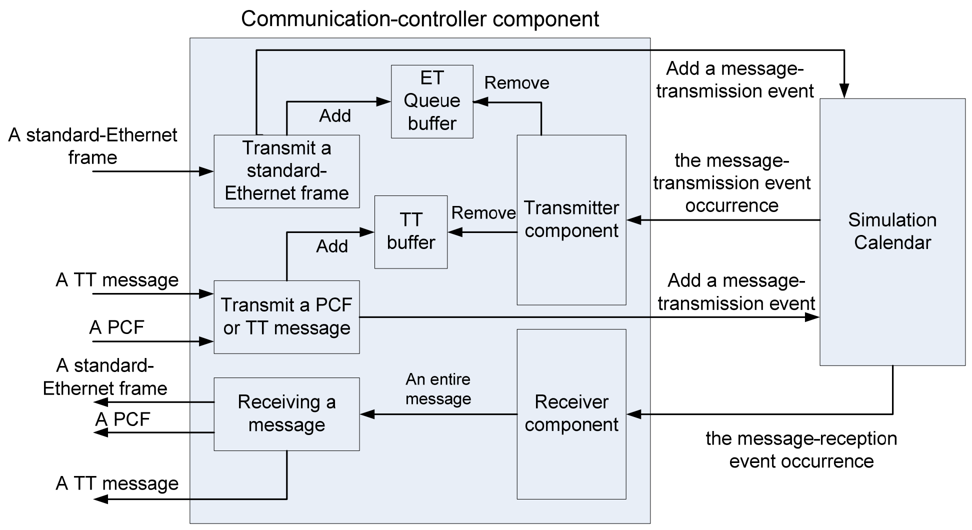

Communication-controller component
ET Queue buffer
Remove
Add
A standard-Ethernet frame
Transmit a standard-Ethernet frame
TT buffer
Remove
Add
Transmitter component
Add a message-transmission event
the message-transmission event occurrence
Simulation Calendar
A TT message
A PCF
Transmit a PCF or TT message
Add a message-transmission event
A standard-Ethernet frame
A PCF
Receiving a message
An entire message
Receiver component
the message-reception event occurrence
A TT message

Fig. 4.10: A *communication-controller* component

A *communication-controller* component has a parameter indicating its bandwidth. In this thesis, we focus on two Ethernet bandwidths: 100 Mbps and 1 Gbps. A *communication-controller* component provides the following methods for handling TT messages, PCFs, and standard-Ethernet frames:

- *Transmitting a standard-Ethernet frame*: This method puts a standard-Ethernet frame into the ET queue buffer of a *communication-controller* component. It then adds a message-transmission event to the simulation calendar. A message-transmission event consists of a *transmitter* component's reference, and the message-transmission event's point in simulation time. When the message-transmission event occurs at its point in simulation time, the *Life* method of the *transmitter* component is processed.

- *Transmitting a PCF or TT message*: This method stores a message from a *TTE-engine* component (i.e. a PCF or TT message) to the TT buffer of a *communication-controller* component. It then adds a message-transmission event to the simulation calendar. A message-transmission event consists of a *transmitter* component's reference, and the message-transmission event's point in simulation time. When this message-transmission event occurs at its point in simulation time, the *Life* method of the transmitter component is processed.

- Receiving a message: This method is invoked when the *receiver* component has received an entire message. If the message is a PCF or TT message, this method will relay it to the *TTE-*

engine component. If the message is a standard-Ethernet frame, this method will relay it to the *Receive* method of the node component.

4.2.6 Transmitter component

A transmitter component is a part of a *communication-controller* component or a *TTE-switch-port* component. It is used for transmitting all messages to the connected *transmission-line* component. All messages to be transmitted come from the ET queue buffer and the TT buffer of an upper component. This upper component is either a *communication-controller* component or a *TTE-switch-port* component. The user-defined bandwidth value of the upper component determines the transmission rate of the *transmitter* component.

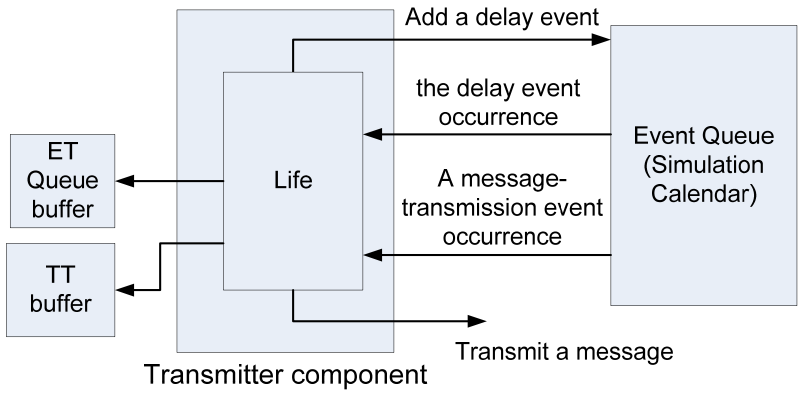

Add a delay event
the delay event occurrence
A message-transmission event occurrence
Event Queue (Simulation Calendar)
ET Queue buffer
TT buffer
Life
Transmit a message
Transmitter component

Fig. 4.11: A *transmitter* component

A *transmitter* component has *Life* methods for transmitting standard-Ethernet frames, PCFs, and TT messages. The *Life* method starts executing when a message-transmission event occurs (from the simulation calendar). The *Life* method has the following processes:

- Check message buffers: Since TT messages and PCFs are prioritized over standard-Ethernet frames, the *Life* method firstly performs the transmission of messages in the TT buffer. If the TT buffer is empty, the transmission of messages in the ET queue buffer is performed.

- Transmit a starting message: We model the transmission simulation of a message by our *two-message transmission* approach, as depicted in Fig. 4.12. The essence of this approach is that one message is split into two messages: *a starting message*, and an *ending message*. A *transmitter* component begins to transmit a message by sending the starting message to the connected transmission-line component.

- Delay transmission time: After the *transmitter* component transmits the starting message, it delays itself with the transmission time of the message. The transmission time is calculated as the length of the entire message divided by the user-defined bandwidth. The *Life* method inserts a delay event into the simulation calendar. The delay event's point in simulation time is equal to the starting-message transmission's point in time plus the delay of the transmission time. When the delay event occurs, the *Life* method is activated to transmit an ending message, as depicted in Fig. 4.12.

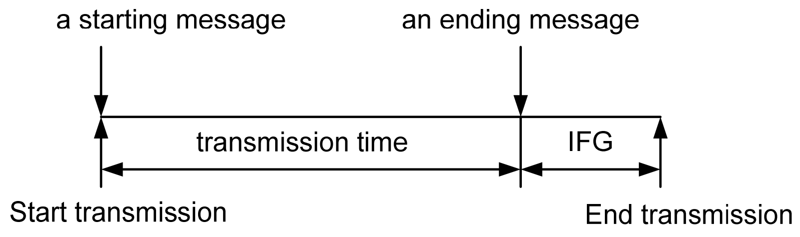


Fig. 4.12: Timing diagram of two-message transmission

- Transmit an ending message: After the delay of the transmission time has elapsed, the ending message is transmitted to the connected transmission-line component.
- Delay with an inter-frame gap: The *Life* method ends by inserting a delay of inter-frame gap (IFG) into the simulation calendar. The delay event's point in simulation time is equal to the ending-message transmission's point in time plus the delay of IFG. When the delay event occurs, the *Life* method is activated to transmit an ending message, as depicted in Fig. 4.12.

The *two-message transmission* approach is useful for identifying an incomplete message, i.e. a receiver component can check if it has received only one of two complementary messages. In the case of transmission of PCFs however, there are a few differences to the steps described above. Before transmitting the starting message of a PCF message, the time interval between the current local-clock time and the dispatch point in local-clock time is accumulated. This is because a PCF may dispatch from a *TTE-engine* component while a *transmitter* component is transmitting a standard-Ethernet frame. Therefore, the PCF has to wait in the TT buffer until the transmission of the standard-Ethernet frame has finished.

4.2.7 Receiver component

A *receiver* component is a part of a *communication-controller* component or a *TTE-switch-port* component. It is used for receiving all messages from the connected *transmission-line* component. The received messages are relayed to the *communication-controller* component or the *TTE-switch-port component*. There are two main methods in a *receiver* component: the *Message-reception* method and the *Life* method. These are described as follows:

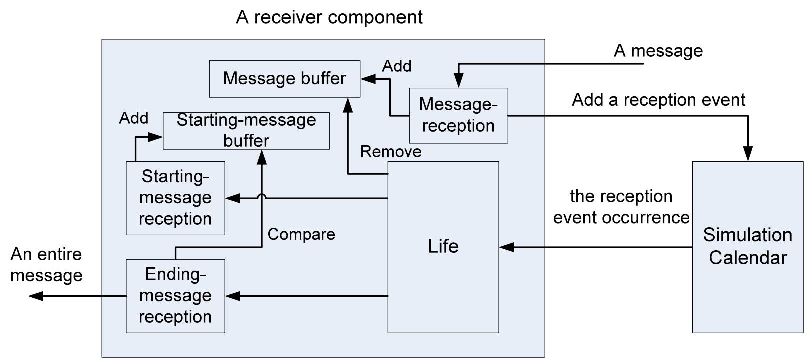


Fig. 4.13: A *receiver* component

- *Message-reception* method: This method is executed when a message is transmitted from the connected *transmission-line* component. The method stores the receiving message to a message buffer, as depicted in Fig. 4.13. The message buffer contains only one message, i.e. storing a message into the message buffer will replace its current message. After a message is stored in the message buffer, a reception event is added to the simulation calendar. A reception event consists of a receiver component reference and the reception event's point in simulation time. The reception event's point in simulation time is equal to the current point in simulation time. This means that a reception event occurs at a receiver component as soon as a message is received. The process of CRC validity is included in this method.

- *Life* method: Once a reception event occurs, the *Life* method is activated to remove the message in the message buffer, as depicted in Fig. 4.13. If the message is a starting message, it will be stored in the *starting-message buffer*. Storing a starting message will replace the current message in the starting-message buffer. This means that there is only one message in the starting-message buffer at any one time. If the received message is an ending message, it is compared with the current message in the starting-message buffer. If it is found that both messages originated from the same message, the ending message is sent to the next process. Otherwise, the ending message is discarded.

4.2.8 TTE-Switch component

In our simulation model, we model a *TTEthernet* switch as a *TTE-switch* component. A *TTE-switch* component is mainly composed of a *TTE-engine* component, one or more *TTE-switch-port* components, and a *local-clock* component, as depicted in Fig. 4.14.

A *TTE-engine* component is responsible for handling PCFs and TT messages, according to the TTEthernet specification [TTE08]. A *TTE-switch* component has the following user-defined parameters:

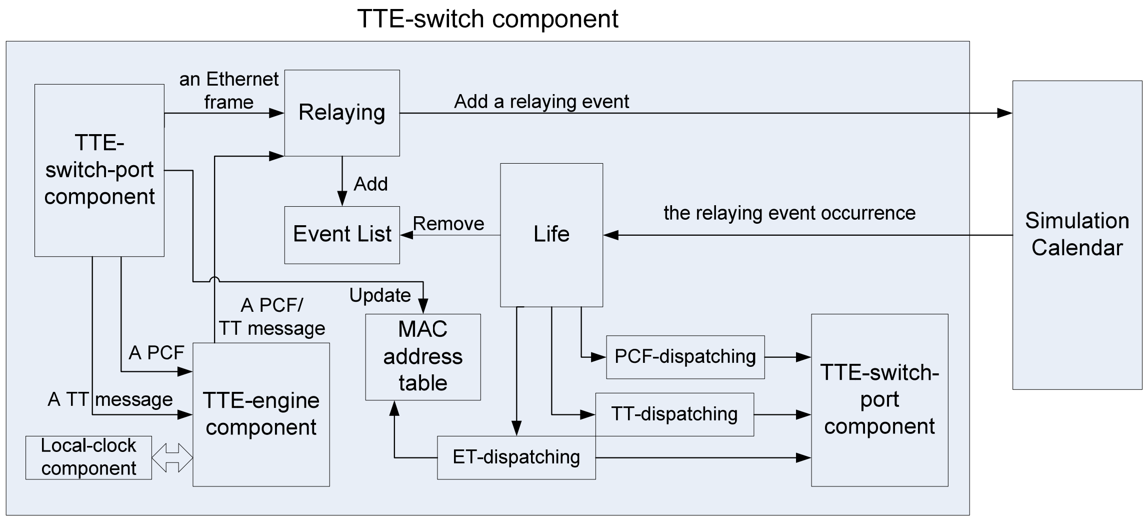


Fig. 4.14: A *TTE-switch* component with two *TTE-switch-port* components

- Buffer Memory Size: The overall buffer memory size of all *TTE-switch-port* components. These buffers are used for storing *Ethernet-frame* components. Due to the non-blocking technology of switched Ethernet, standard Ethernet frames are not queued at an input port of an Ethernet switch. Therefore, we provide the overall buffer memory of a *TTE-switch* component for the queue buffers of all output ports. To efficiently utilize the buffer memory of a TTE switch for the queue buffer, we define the buffer memory of a *TTE-switch* component as *shared buffer memory*. In cases where all standard Ethernet framers are relaying at only a particular *TTE-switch-port* component, that particular component can use all shared buffer memory. In our simulation model, there is no component for shared buffer memory, instead we use a variable indicating the shared-buffer-memory size. Each time a standard Ethernet message is to be stored in the ET buffer of a *TTE-switch-port* component, that component calculates the remaining shared-buffer-memory size. The remaining shared-buffer-memory size ($\text{Size}_{\text{remaining\_sbm}}$) is calculated using Equation 4.5. It is equal to the overall shared-buffer-memory size (Size_{sbm}) minus the currently used shared-buffer-memory size (of all *TTE-switch-port* components).

$$Size_{remaining\_sbm} = Size_{sbm} - \sum_{i=0}^{\pi} size_{ET\_Buffer(i)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(4.5)$$

, where $\text{Size}_{\text{remaining\_sbm}}$ is the remaining shared-buffer-memory size of a *TTE-switch* component

Size_{sbm} is the shared-buffer-memory size of a *TTE-switch* component

$\text{Size}_{\text{ET\_Buffer(i)}}$ is the ET buffer of the i^{th} *TTE-switch-port* component

If the remaining shared-buffer-memory size is less than the size of a received standard-Ethernet frame, the received frame will be stored in the buffer of the *TTE-switch-port* component. Otherwise, the received frame is discarded and memory overflow information is recorded.

- Drift Rate: The local-clock drift rate of a *TTE-switch* component. The *local-clock* component of a *TTE-switch component* is identical to the *local-clock* component of a *node component*.

- ET Latency: The *average delay* when an entire standard Ethernet frame travels from a *receiver* component to the ET buffer of a *transmitter* component (in a TTE-switch).

- TT Latency: The *bounded delay* when an entire TT message frame travels from a *receiver* component to the TT buffer of a *transmitter* component (in a *TTE-switch*).

- Startup Time: Determines a starting point for a *TTE-switch* component in simulation time. If the progression of simulation time has not yet reached the startup-time, all operations of the *TTE-switch* component, and its underlying components (e.g. *local-clock, receiver, transmitter, message-permanence* components) cannot be performed.

A *TTE-switch* component has the following methods for handling all incoming and outgoing messages using its underlying *TTE-switch-port* components:

- *Life*: Executed when a relaying event's point in simulation time (in the simulation calendar) is reached. A relaying event consists of a *TTE-switch* component's reference, and the relaying event's point in simulation time. This method removes the first relaying event in the message list of the *TTE-switch* component. This is the event with the lowest point in simulation time. Depending on the type of message removed, there are three distinct methods used for their dispatching: *ET-dispatching, TT-dispatching, and PCF-dispatching*. These methods are described as follows:

- *ET-dispatching:* Forwards a standard-Ethernet frame (an *ET-frame* component) from the message list of a *TTE-switch* component to its *transmitter* components (in compliance with the standard-Ethernet specification). Firstly, the destination type of the *Ethernet-frame* component is checked (it will be either: broadcast, multicast, or unicast). If this indicates either broadcast or multicast, the *Ethernet-frame* component will be duplicated and then relayed to all other *TTE-switch-port* components. If the *destination type* parameter of the received *Ethernet-frame* component indicates a unicast address, the *receiver* parameter of the received *Ethernet-frame* component is looked up in the *MAC address table*. The MAC address table contains sets of a *TTE-switch-port* component's reference and the corresponding *node* component's reference. If the *receiver* parameter of the received *Ethernet-frame* component matches with the one in the MAC address table, its matching *TTE-switch-port* component is used to relay a duplicated *Ethernet-frame component*. If no matching parameter is found in the MAC address table, this process will relay the received Ethernet-frame component in broadcast form.

- *PCF-dispatching*: Relays a received PCF (a *PCF* component) from the message list of a *TTE-switch* component onto all its other *TTE-switch-port* components. The received *PCF* component is duplicated and then relayed to the *TTE-switch-port* components. Note that all received *PCF* components in the message list are obtained from the *TTE-engine* component. The *TTE-engine* component handles the *PCF* components according to the TTEthernet specification [TTE08].

- *TT-dispatching*: Relays a received TT message (a *TT-message* component) from the message list of a *TTE-switch* component to predefined *TTE-switch-port* components. A received *TT-message* component is duplicated and then relayed to predefined *TTE-switch-port* components. The *transmitter* component of each *TTE-switch-port* component is then activated to transmit the *TT-message* components.

- *Message-relaying*: Delays a message for a user-defined latency. For a standard-Ethernet frame, this method delays an *Ethernet-frame* component for the ET-latency of a *TTE-switch* component. For a PCF/TT message, this method delays a *PCF* or *TT-message* component for the TT-latency of a *TTE-switch component*. This method starts its operation by adding an incoming message and its relaying event's point in simulation time into the message list of a TTE-switch component. The message list contains sets of a message component and its relaying event's point in simulation time. The relaying event's point in simulation time is derived from the point in simulation time when the *message-relaying* method is invoked to relay the message, plus the value of the user-defined latency (i.e. an *ET-latency* value for a standard-Ethernet frame, or a *TT-latency* value for a PCF/TT message). A relaying event is then generated, consisting of a *TTE-switch* component's reference, and the relaying event's point in simulation time. This method cancels the relaying event in the simulation calendar, and then reinserts the relaying event generated from the first message in the message list. Note that all messages in the message list are arranged in ascending order according to their points in simulation time.

4.2.9 TTE-Switch-Port component

We model the communication controller of a TTE switch as a *TTE-switch-port* component. The *TTE-switch-port* component transmits and receives all messages from a connected *transmission-line* component. A TTE-switch-port component is mainly composed of two components: a *transmitter* component, and a *receiver* component. A *TTE-switch-port* component has a queue buffer for standard-Ethernet frames, and a buffer for PCF and TT messages. These are described as follows:

- A queue buffer for standard Ethernet frames (ET queue buffer): This FIFO queue buffer is used for storing *Ethernet-frame* components. The size of the ET queue buffer (in bytes) is defined by the user. The buffer is divided into two sub queue buffers: RC queue buffer and BE queue buffer. An *Ethernet-frame* component has a *Traffic Type* parameter, identifying which sub queue buffer should be used to store it. Removing an *Ethernet-frame* component from the ET queue buffer is to remove the first *Ethernet-frame* component from the RC queue buffer. However, if the RC queue buffer is empty, then the first Ethernet-frame component in the BE queue buffer is read.

- A buffer for PCF or TT messages (TT buffer): This buffer stores a PCF or TT message.

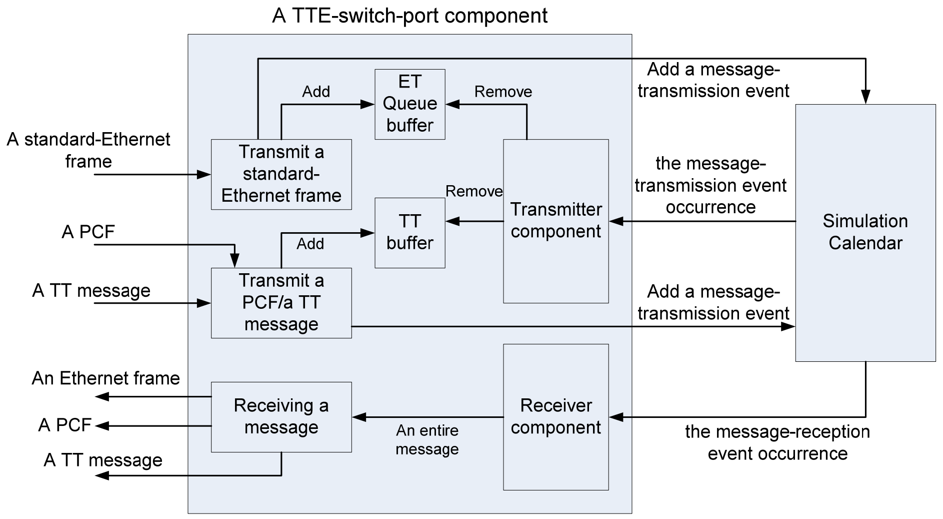

A TTE-switch-port component
A standard-Ethernet frame
A PCF
A TT message
An Ethernet frame
A PCF
A TT message
Transmit a standard-Ethernet frame
Transmit a PCF/a TT message
Receiving a message
Add
ET Queue buffer
Add
TT buffer
Remove
Remove
Transmitter component
Receiver component
An entire message
Add a message-transmission event
the message-transmission event occurrence
Add a message-transmission event
the message-reception event occurrence
Simulation Calendar

Fig. 4.15: A *TTE-switch-port* component

A *TTE-switch-port* component has a parameter indicating its bandwidth. In this thesis, we focus on two Ethernet bandwidths: 100 Mbps and 1 Gbps. A TTE-switch-port component provides the following methods for handling TT messages, PCFs, and standard-Ethernet frames:

- Transmitting a standard-Ethernet frame: This method puts an *Ethernet-frame* component to the ET queue buffer of a TTE-switch-port component. It then adds a message-transmission event to the simulation calendar. A message-transmission event consists of a transmitter component's reference, and the message-transmission event's point in simulation time. When the message-transmission event occurs at its point in simulation time, the *Life* method of the *transmitter* component is processed. Note that every time an *Ethernet-frame* component is stored in the ET queue buffer of a *TTE-switch-port* component, the method checks if the remaining shared-buffer-memory size of the *TTE-switch* component is sufficient (as described in the section 4.2.9). If not, the *Ethernet-frame* component will not be stored in the ET queue buffer, and buffer overflow information is recorded.

- Transmitting a PCF or TT message: This method stores a message from a *TTE-engine* component (i.e. a PCF or TT message) to the TT buffer. It then adds a message-transmission event to the simulation calendar. A message-transmission event consists of a *transmitter* component's reference, and the message-transmission event's point in simulation time. When this message-transmission event occurs at its point in simulation time, the *Life* method of the *transmitter* component is processed. Once the *local-clock* component of a *TTE-engine* component is globally synchronized, the TT buffers of all underlying *TTE-switch-port* component are emptied.

- Receiving a message: This method receives an entire message from the *receiver* component, as depicted in Fig. 4.15. If the message is a *PCF* or *TT-message* component, it is forwarded to the *TTE-engine* component. If the message is an *Ethernet-frame* component, it is forwarded to the *Message-relaying* method of a *TTE-switch* component. Before forwarding the *Ethernet-frame* component, the MAC address table is queried for the *sender* parameter of the *Ethernet-frame*. A check is made to see if the *sender* parameter matches with any node's reference. If there is not a match, the *sender* parameter of the *Ethernet-frame* component and the *TTE-switch-port* component are recorded as the *node* component's reference and the *TTE-switch-port* component's reference (in the MAC address table). If there is a match, a further check is made between the *TTE-switch-port* component of the matching node in the MAC address table and the *TTE-switch-port* component which the *Ethernet-frame* component is from. If both *TTE-switch-port* components are not matched to each other, the *TTE-switch-port* component's reference in the MAC address table is replaced with the one from the *Ethernet-frame*. If a match is found in the MAC address table, the MAC address table remains unmodified.

4.2.10 TTEthernet-Engine (TTE-Engine) component

We model a *TTEthernet* engine as a *TTE-engine* component, as shown in Fig. 4.16. There are four main components inside a *TTE-engine* component. These are described as follows:

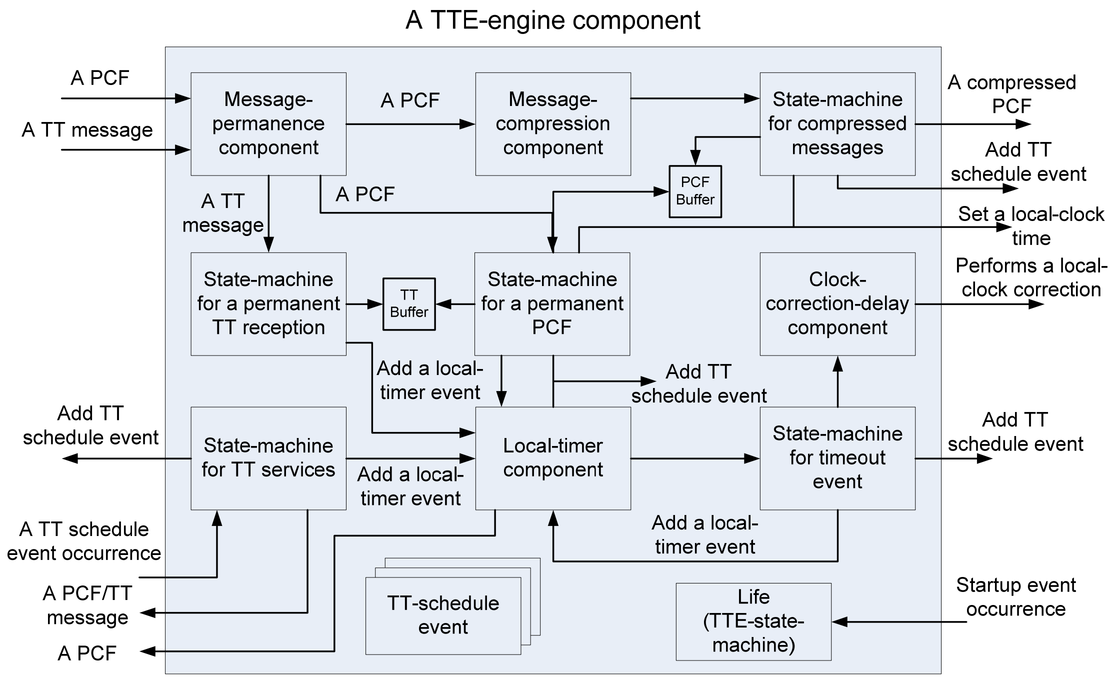


Fig. 4.16: A *TTE-engine* component

1. ***Message-permanence* component:** This component delays an incoming message (a *PCF* component) until its permanent point in simulation time (i.e. until the message has become permanent). This *PCF* component is forwarded from a *communication-controller* component or *TTE-switch-port* component. In a *message-permanence* component, there is a PCF List and the two main methods. These are described as follows:

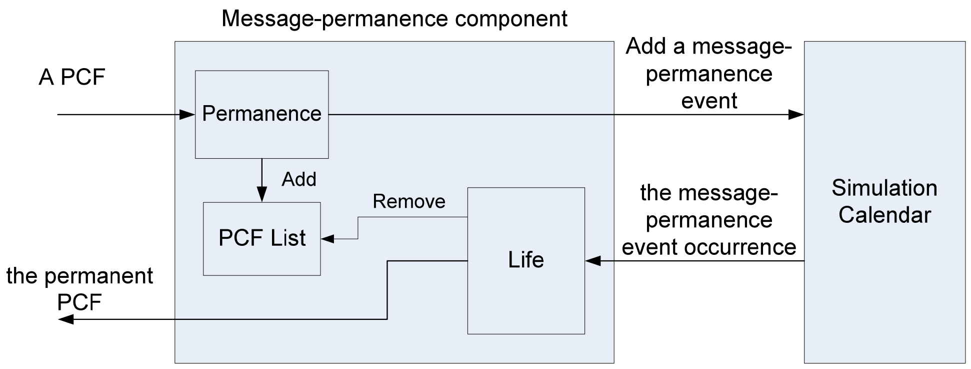

Message-permanence component
A PCF
Permanence
Add
PCF List
Remove
Life
the permanent PCF
Add a message-permanence event
the message-permanence event occurrence
Simulation Calendar

Fig. 4.17: A message-permanence component

- *Permanence* method: This method receives an incoming *PCF* component from a *communication-controller* component or a *TTE-switch-port* component. A message-permanence event is generated, consisting of the message-permanence component's reference and the *PCF* component's calculated permanence point in simulation time. The message-permanence event is then added into the simulation calendar. A message-permanence event consists of the *message-permanence* component's reference, and the message-permanence event's point in simulation time. The *PCF* component is also added into the PCF list of the *message-permanence* component. The permanence delay of a PCF is obtained by the following process.

  Permanence delay of a PCF: This process determines a permanence point in simulation time for a received *PCF* component. Once a *TTE-engine* component has received a *PCF* component, the time interval between the receiving time instant of the starting message and the receiving time instant of the corresponding ending message is accumulated in the transparent clock (of the *PCF* component). Note that this time interval is equal to the transmission time of the *PCF* component, as illustrated in Fig. 4.11. Besides the time interval, the predefined propagation delay of the connected *transmission-line* component is also accumulated in the transparent clock of the *PCF* component. The permanent delay of the *PCF* component is obtained using Equation 4.6. The permanence delay determines the permanence point in simulation time, which is derived from Equation 4.7.

$T_{permanence\_delay} = T_{max\_transmission\_delay} - T_{transparent\_clock}$ ……………………………………(4.6)

$T_{permanence\_pit} = T_{current\_sim} + T_{permanence\_delay}$ ……………………………………………..(4.7)

, where $T_{permanence\_delay}$ = the permanence delay of a *PCF* component

$T_{max\_transmission\_delay}$ = the maximum transmission delay of a TTE cluster

$T_{transparent\_clock}$ = the transparent clock of a *PCF* component

$T_{current\_sim}$ = a current point in simulation time

$T_{permanence\_pit}$ = a permanence point in simulation time

- *Life* method: This method forwards a permanent PCF to the permanent-message service in a *TTE-engine* component. This occurs at the same time as a message-permanence event. A permanent *PCF* component from a *message-permanence* component is forwarded to either: a *message-compression* component, or the TTE state machine for a permanent PCF. If a *TTE-engine* component is identified as a compression master, permanent *PCF* components are forwarded to the *message-compression* component, as shown in Fig. 4.16. Otherwise, the permanent *PCF* components are forwarded to the *TTE state machine for permanent PCFs*. The TTE state machine for permanent *PCFs* provides functions to handle the arrival of permanent PCFs.

2. ***Message-compression* component:** We model the compression function of TTEthernet as the *message-compression* component. This component delays a permanent *PCF* component until its compressed point in time. There are two main methods in a compression component: *Compression* and *Life*. These are described as follows:

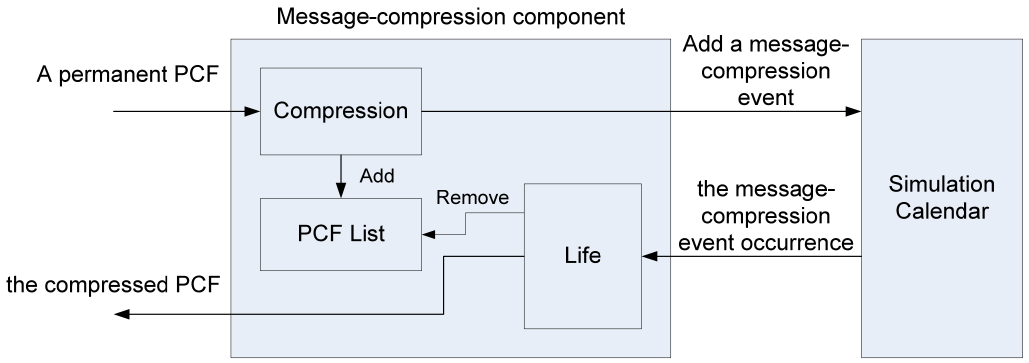

Message-compression component
A permanent PCF
Compression
Add
Remove
PCF List
Life
the compressed PCF
Add a message-compression event
the message-compression event occurrence
Simulation Calendar

Fig. 4.18: A message-compression component

- *Compression*: Determines a permanent PCF's compressed point in simulation time, as derived from the compression function of TTEthernet [TTE08]. Firstly, a message-compression event is generated, consisting of the *message-compression* component's reference, and the permanent PCF's compressed point in simulation time. This message-compression event is added into the simulation calendar. A permanent *PCF* component is then added into the PCF list of the message-compression component, as depicted in Fig. 4.18.

- *Life*: Forwards a *TTE-engine* component's permanent message from the PCF list to the *TTE state machine for message compression*. This occurs at the same time as a message-compression event, meaning that the message-compression event's compressed point in simulation time has been reached. After the *TTE state machine for message compression* has been executed, the compressed *PCF* component is dispatched at the dispatch point in time, as depicted in Fig. 4.19. A compressed PCF's dispatch point in time is derived from its compressed point in time plus a user-defined dispatch delay. In our simulation model, a compressed *PCF* component is dispatched from a *TTE-engine* component to a *communication-controller* component (in a *node* component), or to a *TTE-switch-port* component (in a *TTE-switch* component), at a user-defined dispatch point in time.

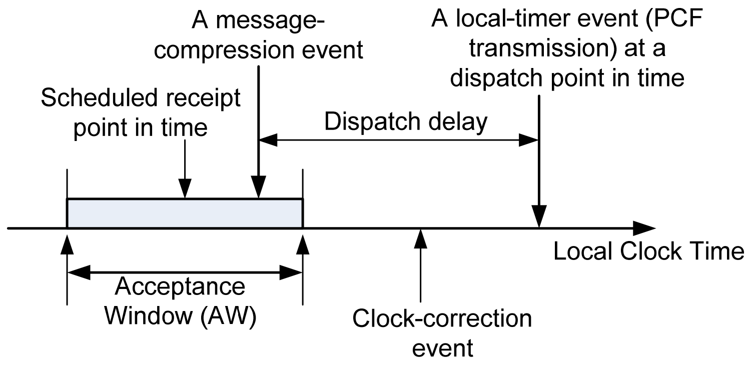

A message-compression event
A local-timer event (PCF transmission) at a dispatch point in time
Scheduled receipt point in time
Dispatch delay
Local Clock Time
Acceptance Window (AW)
Clock-correction event

Fig. 4.19: Timing diagram for PCF transmission in a compression master

3. ***Local-timer* component:** We model a timer for a TTEthernet engine as a local-timer component, as depicted in Fig. 4.20. The TTE state machines in a TTE-engine component can obtain a timer from a local-timer component by invoking the *start-timer* method. The *start-timer* method begins by stopping the currently running timer, canceling the local-timer event in the simulation calendar. A new local-timer event is then added into the simulation calendar. A local-timer event consists of the *local-timer* component's reference, and the local-timer event's point in simulation time. This point in simulation time is a timeout event's point in simulation time. It is derived from Equation 4.8.

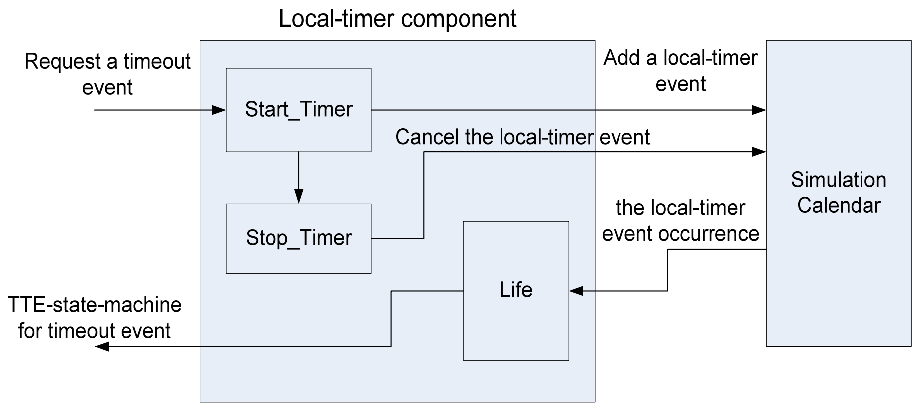


Fig. 4.20: A local-timer component

Tsim_timeout = Tsim_current + TItiming/(1+ρ) ……………..…………………….…….….(4.8)

, where Tsim_timeout = a timeout event's point in simulation time

Tsim_current = a current point in simulation time

TItiming = a time interval of a timer

ρ = the drift rate of a local clock

Once the timeout event's point in simulation time is reached, the *Life* method is activated to execute the *TTE state machine for timeout event*. The TTE state machine for timeout event is a TTE state machine concerning timeout event occurrence. Examples of the timeout parameters of TTEthernet [TTE08] are as follows: cm_listen_timeout, sm_coldstart_timeout, sm_restart_timeout, sm_listen_timeout, cm_ca_timeout, sm_ca_offset, and sm_cs_offset.

4. **Clock-correction-delay component:** We model a TTEthernet clock correction delay as a *clock-correction-delay* component, as depicted in Fig. 4.21.

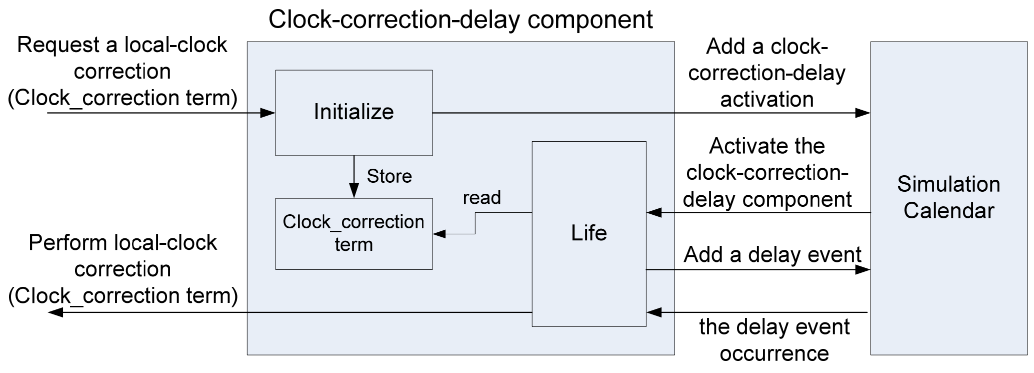

Clock-correction-delay component
Request a local-clock correction (Clock_correction term)
Initialize
Store
Clock_correction term
read
Life
Perform local-clock correction (Clock_correction term)
Add a clock-correction-delay activation
Activate the clock-correction-delay component
Add a delay event
the delay event occurrence
Simulation Calendar

Fig. 4.21: A *clock-correction-delay* component

A *TTE-engine* component requests a local-clock time correction through its *clock-correction-delay* component. This request is triggered by a timeout event of the *local-timer* component, as depicted in Fig. 4.22. In Fig. 4.21, a *clock-correction-del*ay component stores the received clock-correction term, and activates the *Life* method. The *Life* method creates a time delay by adding a delay event to the simulation calendar. A delay event consists of a *clock-correction-delay* component's reference, and the delay event's point in simulation time. This point in simulation time can be calculated using Equation 4.9.

$T_{sim\_delay\_event} = T_{sim\_current} + TI_{time\_delay}/(1+\rho)$ ……………………………………...…...…(4.9)

, where $T_{sim\_delay\_event}$ = a delay event's point in simulation time

$T_{sim\_current}$ = a current point in simulation time

TI_{time_delay} = a time delay based on local-clock time

ρ = the drift rate of a local clock

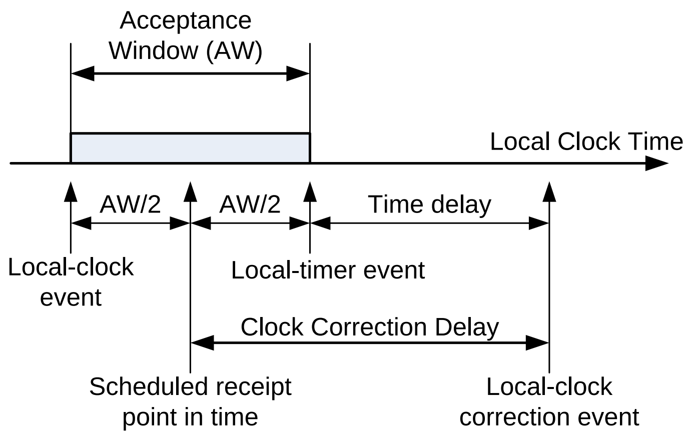


Fig. 4.22: Timing diagram of a local-clock correction event

The time delay for a delay event is derived from a clock correction delay minus half of the acceptance window, as depicted in Fig. 4.22. The clock correction delay is a user-defined configuration parameter of TTEthernet. When the delay event's point in simulation time is reached, the *Life* method of the clock-correction-delay component invokes the *local-clock* component to correct its local-clock time.

Time-triggered communication services

A *TTE-engine* component performs time-triggered communication services by using the *local-clock* component within the same *node* or *TTE-switch* component. The *TTE-engine* component requests a time-triggered communication service by adding a time-triggered schedule event to the *local-clock* component, as depicted in Fig. 4.16 and Fig. 4.8. A time-triggered schedule event consists of a TT communication service, and the TT schedule event's point in simulation time. When a time-triggered schedule event's point in simulation time is reached, the specified TT communication service is performed. A *TTE-engine* component provides the *TTE state machine for TT communication services* to achieve this. These TT communication services are described as follows:

- ✓ SM_DISPACTH_SERVICE: Used for dispatching an integration frame (PCF) at a synchronization master or synchronization client.
- ✓ SMC_SCHEDULED_RECEIVICE_SERVICE: Used for receiving an integration frame (PCF) at a synchronization master or synchronization client.
- ✓ CM_SCHEDULED_RECEIVE_SERVICE: Used for receiving an integration frame (PCF) at a compression master.
- ✓ SMC_ASYNC_UPDATE_SERVICE: Used for updating the membership state at a synchronization master or synchronization client.
- ✓ CM_ASYNC_UPDATE_SERVICE: Used for updating the membership state at a compression master.
- ✓ SMC_ASYNC_EVAL_SERVICE: Used for the asynchronous clique detection of a synchronization master or synchronization client.
- ✓ TT_DISPATCH_SERVICE: Used for dispatching a TT message.
- ✓ TT_SCHEDULE_RECEIVE_SERVICE: Used for receiving a TT message with regard to a TT communication schedule.

TTE state machines

In our simulation model, we split the TTE state machine of TTEthernet into several event occurrence specific TTE state machines. These TTE state machines belong to a *TTE-engine* component. In a *TTE-engine* component, we use a *global state* variable to determine the current state of all TTE state machines. We call this global state variable a *protocol state variable*. All TTE state machines use the same protocol state variable to determine which state's process has to be performed, as depicted in Fig. 4.23.

- ➢ TTE state machine for a permanent PCF: This TTE state machine is invoked when a *PCF* component has become permanent at the permanence point in time. This is invoked if and only if the *TTE-engine* component is a synchronization master or a synchronization client.

- TTE state machine for a compressed PCF: Invoked when a compressed *PCF* component arrives at a compressed point in time.
- TTE state machine for a timeout event: Invoked when a timeout event of a *local-timer* component occurs.
- TTE state machine for TT reception: Invoked when an entire *TT-message* component is received.
- TTE state machine for TT services: Invoked when a time-triggered schedule event from a *local-clock* component occurs. Each state of this TTE state machine has methods for the respective TT schedule services, as mentioned in the *time-triggered communication services* section.
- TTE state machine for the *Life* method: Used by the main method of a *TTE-engine* component to interface with the simulation calendar. There is only the *Init* state in this method. An initialized *TTE-engine* component takes the *Init* state, where the TTEthernet configuration (including a user-defined synchronization device type) and TT communication schedule are loaded.

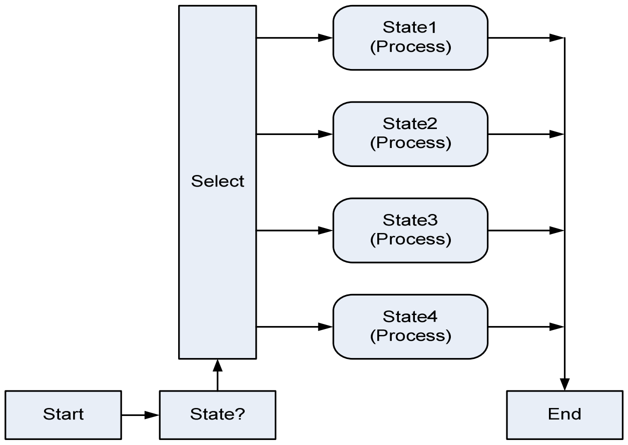


Fig. 4.23: The operation model of a TTE state-machine

Message Transmission

With respect to the globally synchronized *TTE-engine* component, both TT-messages and PCFs are transmitted according to the TT communication schedule. If the *TTE-engine* component is not globally synchronized, only PCFs are transmitted according to the TTE state machines. A *TTE-engine* component transmits TT messages and PCFs by dispatching them to a *TTE-switch-*

port component or a *communication-controller* component. Note that no standard-Ethernet frames are forwarded from a *TTE-engine* component. The transmission of TT messages and PCFs are described with regard to synchronization device type as follows:

➢ Dispatching PCFs: In a *TTE-engine* component, PCF transmissions happen in a synchronization master and a compression master. There are no PCF transmissions in a synchronization client.

- Synchronization master: A *TTE-engine* component transmits PCFs with regard to synchronization state. If the *TTE-engine* component is not in a synchronization state, the *TTE-engine* component sends PCFs by means of the *Fault-tolerance Handshake* [TTE08]. This is where timeout events are used for determining a PCF's dispatching point in simulation time. If the *TTE-engine* component is in a synchronization state, the *TTE-engine* component sends PCFs (only integration frames) according to the TT communication schedule. That is where the *TTE-engine* component requests a TT schedule event, including the TT communication service (SM_DISPACTH_SERVICE) at the *local-clock* component.

- Compression master: A *TTE-engine* component transmits a compressed PCF at the compressed PCF's dispatch point in time, as depicted in Fig. 4.24. The dispatch point in time is derived from the compressed PCF's compressed point in time plus a user-defined dispatch delay. In our simulation model, once a message-compression event has occurred, the TTE state machine for a compressed message requests a local-timer event for dispatching the compressed PCF. This local-timer event has the same timeout value as the user-defined dispatch delay.

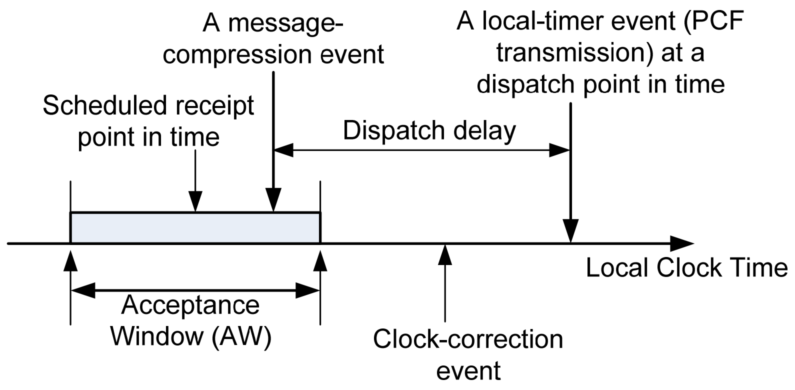

A message-compression event
A local-timer event (PCF transmission) at a dispatch point in time
Scheduled receipt point in time
Dispatch delay
Local Clock Time
Acceptance Window (AW)
Clock-correction event

Fig. 4.24: Timing diagram for compressed-PCF transmission

➢ Dispatching TT messages: The *TTE-engine* component transmits TT messages according to the TT communication schedule, if it is in a synchronization state. The *TTE-engine* component requests a TT schedule event from the *local-clock* component. This includes a TT communication service (TT_DISPATCH_SERVICE).

4.2.11 TT-schedule component

A *cluster* component has the *cluster-TT-schedule* list used for storing *TT-schedule* components. A *TT-schedule* component contains a user-defined TT-traffic load. This TT-traffic load has the following parameters:

- Cluster: Indicates the *cluster* component.
- Node: Indicates the *node* component.
- In/Out: Indicates a TT message direction: transmission or reception.
- Period: The assigned period.
- Period_ID: Identification of the assigned period.
- Size: The TT-message size.
- Phase: The offsets of the assigned period, obtained by our scheduling approach.

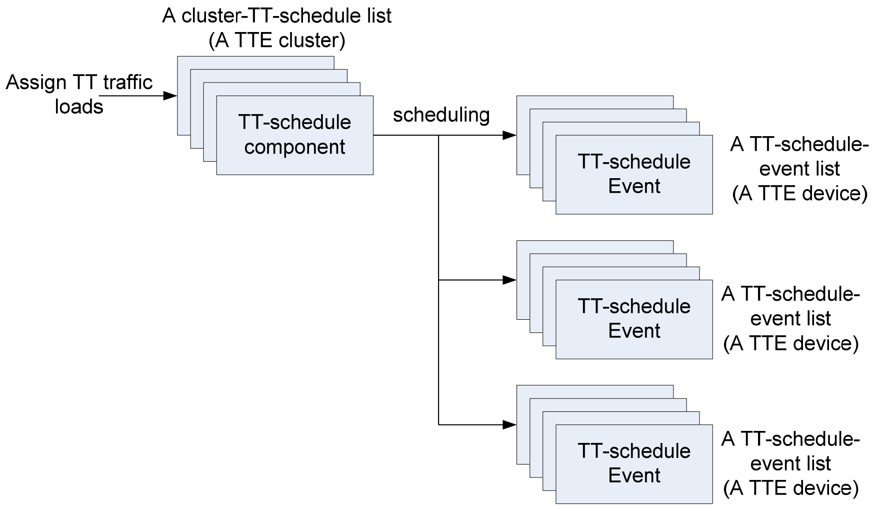

A cluster-TT-schedule list
(A TTE cluster)
Assign TT traffic loads
TT-schedule component
scheduling
TT-schedule Event
A TT-schedule-event list (A TTE device)
TT-schedule Event
A TT-schedule-event list (A TTE device)
TT-schedule Event
A TT-schedule-event list (A TTE device)

Fig. 4.25: The *TT-schedule* components of a *cluster* component (three TTE devices)

A *TTE-engine* component has a *TT-schedule-event* list that contains all TT-schedule events in a TTE cluster cycle (as depicted in Fig. 4.25). We model a TT-message communication event as a TT-schedule event, consisting of a TT-schedule component, and the TT-schedule event's point in time. All TT-schedule events in a *TT-schedule-event* list are in ascending order with respect to their points in time. All the TT-schedule event's points in time for a TTE cluster cycle are obtained by our scheduling approach.

4.2.12 TT-message component

We model a TT message as a *TT-message* component. The *TT-message* component has the following parameters:

- Period_ID: Identifies the user-assigned TT-traffic load of the *TT-message* component. This parameter identifies the *TT-schedule* components that the *TT-message* component belongs to.
- Data: Contains the payload data of the *TT-message* component.
- Data Length: Determines the payload data length.
- Sender: Identifies the *node* component which transmits the *TT-message* component.
- Frame Check Sequence: A CRC value contained in the *Frame Check Sequence* field of the standard-Ethernet frame (upon which the *TT-message* component is based).

4.2.13 PCF component

We model a PCF as a *PCF* component. The *PCF* component has the following parameters:

- Integration Cycle: The sequence number of the *PCF* component in a TTE cluster cycle. Used in a PCF as an integration frame.
- Membership: Contains the membership value of a synchronization device. If this parameter is sent from a synchronization master device, it is a unique membership of this device. If however this parameter is sent from a compression master device, it is a membership result from the compression function of TTEthernet.
- PCF Type: Indicates the type of the *PCF* component. There are three types of PCFs: Integration frames, coldstart frames, and coldstart acknowledge frames.
- Transparent Clock: Contains the accumulated transport delay of the *PCF* component.
- Frame Check Sequence: A CRC value contained in the *Frame Check Sequence* field of the standard-Ethernet frame (upon which the PCF component is based).

4.2.14 Ethernet-frame component

We model a standard-Ethernet frame as an *Ethernet-frame* component. The *Ethernet-frame* component has the following parameters:

- Sender: Indicates the *node* component which transmits the *Ethernet-frame* component.

- Receiver: Indicates the *node* component which is predefined to receive the Ethernet-frame component.

- Destination Type: Indicates the type of the *Ethernet-frame* component. There are three types of standard-Ethernet frames: unicast, multicast, and broadcast.

- Data: Contains the payload data.

- Data Length: Determines the payload data length.

- Traffic Type: Used for determining which sub queue buffer of the ET buffer (in a *node* or *TTE-switch* component) the *Ethernet-frame* component has to be stored. There are two possible parameter values: RC traffic and BE traffic.

- Frame Check Sequence: A CRC value contained in the *Frame Check Sequence* field of the *Ethernet-frame* component.

4.2.15 TTE-configuration component

We provide a *TTE-configuration* component for containing all synchronization parameters of TTEthernet. A TTE-configuration component is part of a *cluster* component. All synchronization parameters are TTE-cluster-specific. These parameters are described as follows:

1. **Maximum transmission delay**: A bounded value of a PCF's accumulated transport delay from one node to another (in a TTE cluster). This parameter is user-defined, and is equal to or greater than the maximum possible value of the accumulated transport delay of a PCF.

2. **Local clock precision**: Denotes a bounded clock precision in the local view of any TTE device in a TTE cluster. Its default value is 1 microsecond.

3. **Acceptance window**: Denotes a bounded clock precision in the global view of any TTE device in a TTE cluster. It is equal to two times the local clock precision of the TTE cluster.

4. **Observation window**: Used in the compression function of a compression master. This value indicates a maximum possible error of arrival for messages in the local view of the compression master. It is equal to the value of the local clock precision in the TTE cluster.

5. **Number of faulty masters:** The upper bound on the permitted number of faulty synchronization masters (in a TTE cluster). Used in the compression function of a compression master. This value determines the maximum observation window of the compression master.

6. **Maximum observation window**: Used in the compression function of a compression master. This value is a result of defining the number of faulty masters and the observation window, as detailed in the TTEthernet specification [TTE08].

7. **Calculation overhead**: A bounded time delay for calculating the function of fault-tolerant clock synchronization (in the compression function of TTEthernet). The result of the fault-tolerant clock synchronization is a local-clock correction value.

8. **Clock correction delay**: A time delay used for determining when, in the process of clock synchronization, should local-clock correction occur. This point in time is equal to a PCF's scheduled reception point in time plus a clock correction delay. This value should be greater than an acceptance window, so that the corrected local-clock value will not occur within the previous acceptance window. Therefore, we define the clock correction delay in the simulation model to be 1.5 times the acceptance window.

9. **Integration cycle duration**: A resynchronization time interval of two consecutive PCFs. This is achieved by all synchronization masters in a particular cluster sending their respective PCF every specified time period. This value is derived from our scheduling approach.

10. **Cluster cycle duration**: The duration of a cluster cycle. In a TTE system, a local clock has a maximum local-clock time of cluster cycle duration. This value is derived from our scheduling approach.

11. **sm_listen_duration**: Used in the TTE state machine, as detailed in the TTEthernet specification [TTE08].

12. **sm_coldstart_timeout**: Used in the TTE state machine, as detailed in the TTEthernet specification [TTE08].

13. **sm_cs_offset**: Used in the TTE state machine, as detailed in the TTEthernet specification [TTE08].

14. **sm_ca_offset**: Used in the TTE state machine, as detailed in the TTEthernet specification [TTE08].

15. **sm_restart_timeout**: Used in the TTE state machine, as detailed in the TTEthernet specification [TTE08].

16. **cm_ca_timeout**: Used in the TTE state machine, as detailed in the TTEthernet specification [TTE08].

17. **cm_in_timeout**: Used in the TTE state machine, as detailed in the TTEthernet specification [TTE08].

4.3 The calibration of the simulation model

We have implemented our simulation model using the *J-Sim* library and the Java run-time environment. We perform the calibrations of the implemented model with published results. In the implemented simulation, there are two types of message traffic: standard-Ethernet traffic and TT traffic. For TT traffic, the TT communication is schedule-predefined and deterministic. Therefore, the validation process of TT communication was performed according to a predefined confliction-free TT-communication schedule. For standard-Ethernet traffic, communication is non-deterministic. The non-deterministic latency of a standard-Ethernet frame is caused by frame queuing in an Ethernet switch. Therefore, we perform the calibrations of the implemented simulation with published results from [LL02] and [Rug08]. This is described in detail in the sections that follow.

4.3.1 Queuing delay of switched Ethernet

We calibrate our implemented simulation with the analytical result from [LL02]. This paper provides an analytical model for the queuing delay of a standard-Ethernet frame, as shown in Equation 4.10 [LL02].

$$D_Q = \sum_{k=1}^{N_q} [96 + max(L_k + L_h, 576)] t_b \text{ ....(4.10)}$$

, where D_Q is the queuing delay which a standard-Ethernet frame takes,

t_b is one-bit time of the Ethernet-frame transmission,

L_h is the bit length of a standard-Ethernet-frame overhead including the frame check sequence,

L_k is the bit length of the payload data in the k^{th} standard-Ethernet frame,

N_q is the number of frames which are contained in the queue buffer.

Note that the smallest number of bits in a standard-Ethernet frame (i.e. for the payload data) is 46 bytes. The t_b is one-bit time of Ethernet-frame transmission (e.g. it is equal to 1/10Mbps for 10-Mbps Ethernet).

According to [LL02], an Ethernet network will obtain the maximum queuing delay from the analytical model. This Ethernet network consists of 14 slave stations, one master station, and one Ethernet switch. All the slave stations generate a standard-Ethernet frame every 1 millisecond, and then transmit it to the master station. The bandwidth of each Ethernet device is 10 Mbps. The payload data size of each Ethernet frame is defined as 144 bits (i.e. this result in 576 bits of a frame size). In a worst-case communication delay (maximum communication delay), a standard-Ethernet frame is received by the Ethernet switch with 13 frames stored in the queue buffer. By means of the analytical model (Equation 4.10), the *maximum queuing delay* of such a standard-Ethernet frame is **873.2 μs** [LL02].

We calibrate the implemented simulation with the result of the analytical model by measuring the *communication jitter* of the Ethernet network. The communication jitter is the difference between the maximum and minimum end-to-end delays. We found that the communication jitter is equal to the *maximum queuing delay* of the Ethernet switch. Communication jitter does not depend on network configuration, implying that it remains constant in any network configuration (e.g. network cable length, switch latency, forwarding latency) except Ethernet bandwidth. In other words, the jitter communication delay is a maximum delay that a standard-Ethernet frame spends in the queue buffer of an Ethernet switch. In the simulation, we record the end-to-end delays of all standard-Ethernet frames (from the slave stations) at the master station. The maximum and minimum end-to-end delay of the standard-Ethernet frames can be obtained from the recorded end-to-end delays. We describe how we setup the network configuration of this Ethernet network below. We define the standard-Ethernet traffic load of this Ethernet network according to the one defined in the analytical model (i.e. all slave stations transmit a 576-bit standard-Ethernet frame every period of 1 millisecond). In this simulation, we define all slave stations to generate their respective Ethernet traffic loads at the same point in simulation time, so that we can obtain the worst-case communication jitter. The Ethernet network structure of this simulation is shown in Fig. 4.26. The results from both this simulation and the analytical model are shown in Table 4.1. We conclude that the simulation communication jitter coincides with the analytical queuing delay (873.6 μs).

Network Configuration: We setup the network configuration with the following parameter values:

- Ethernet Switch: The configuration of the Ethernet switch is based on the following *3COM SuperStack II Switch 1100* specification:
    - Bandwidth: 10 Mbps
    - Shared Buffer Memory: 800 Kbytes (dynamic TX shared buffer memory)
    - Ethernet Latency (ET_Latency): 8 microseconds (Store and Forward)
- Ethernet Controller: 10 Mbps
- Network Cable: 0.1 μs. This is derived from the propagation speed of 2.0 x 10^8 m/s and a cable length of 20m.

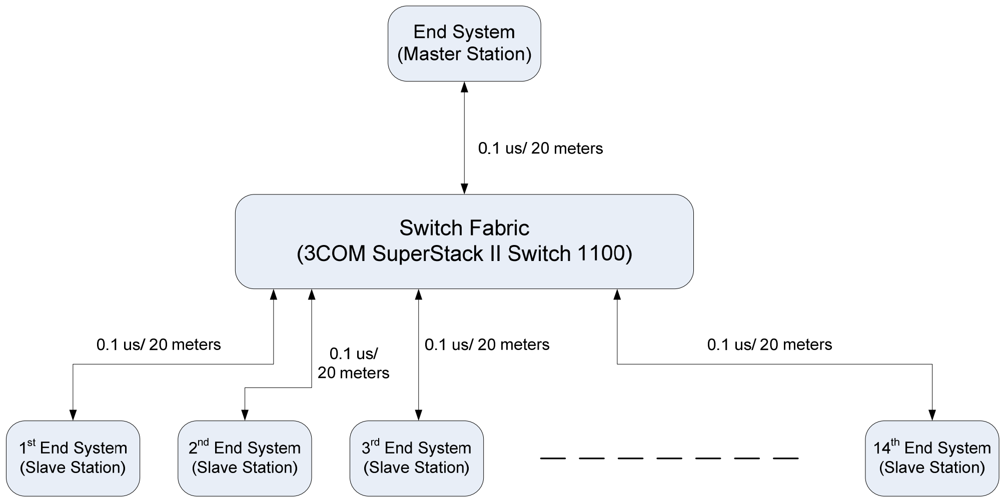


Fig. 4.26: The network configuration, 14 slave stations and one master station

Table 4.1: Maximum, minimum, and jitter of the end-to-end delay from the slave stations to the master station

Frame size (bits)	Our simulation model			Analytical model
	Maximum end-to-end delay (µs)	Minimum end-to-end delay (µs)	Jitter (µs)	Queuing delay (µs)
576	997.833	124.233	**873.6**	**873.6**

4.3.2 Latency on a switched Ethernet network

We calibrate our implemented simulation with the analytical results from [Rug08]. This paper provides an analytical model for the switch latency of a standard-Ethernet frame (in an Ethernet switch). The switch latency is a time interval. It starts from the point when an entire standard-Ethernet frame arrives at an input port, and ends at the point when the last bit of that frame is transmitted from an output port. In cases where several Ethernet switches coexist, the switch latency is the accumulated latency for all Ethernet switches where the Ethernet frame has passed. In this paper there are four main scenarios which each have analytical results. These scenarios are explained in the sections that follow.

4.3.2.1 Simple Ethernet Network: This scenario considers a simple network consisting of several nodes connected to an Ethernet switch. In this scenario there are 15 transmitter nodes and one receiver node, as shown in Fig. 4.27.

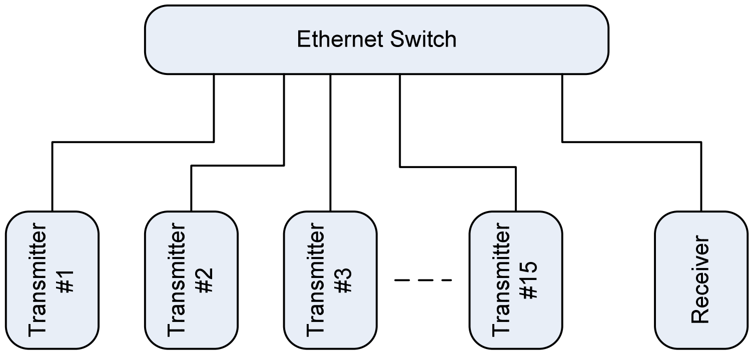


Fig. 4.27: A simple Switched Ethernet Network

All Ethernet-controller bandwidths in this network are 100 Mbps. The switch fabric latency of an Ethernet switch is the same as that of a *RuggedSwitch* product (i.e. 5.2 µs) [Rug08]. This scenario is divided into the following three sub-scenarios:

- Light network load with a minimum-size frame of 64 bytes: In this scenario, one transmitter node transmits standard-Ethernet frames to the receiver node. The lengths of these standard-Ethernet frames are identical and equal to *64 bytes*. In our simulation, we define the transmission period of the standard-Ethernet frames as 1 millisecond. The results from our implemented simulation and the analytical model are shown in Fig. 4.28.
- Light network load with a maximum-size frame of 1518 bytes: In this scenario, one transmitter node transmits standard-Ethernet frames to the receiver node. The lengths of these standard-Ethernet frames are identical and equal to *1518 bytes*. In our simulation, we define the transmission period of the standard-Ethernet frames as 1 millisecond. The results from our implemented simulation and the analytical model are shown in Fig. 4.29.
- Heavy network load with a maximum-size frame of 1518 bytes: In this scenario, every transmitter node transmits standard-Ethernet frames to the receiver node. The lengths of these standard-Ethernet frames are identical and equal to *1518 bytes*. The analytical model considers the worst-case switch latency, where the Ethernet switch receives frames from every transmitter node at the same point in time. The worst-case switch latency is a time interval between when the Ethernet switch starts to receive the last frame, to when it starts to transmit it. The results from our implemented simulation and the analytical model are shown in Fig. 4.29.

4.3.2.2 IEC 61850-9-1 Traffic: IEC 61850 is a standard for the design of electrical substation automation. The communication protocol of IEC 61850 is defined as high-speed switched Ethernet. This scenario has a network consisting of 12 transmitter nodes, one receiver node, and one Ethernet switch. The data in each transmitter node is sampled with a sampling rate of 128 samples per cycle (60 Hz). This means that the transmission period of these standard-Ethernet frames is 1/(128*60) seconds. Each standard-Ethernet frame has a bit-length of 984 bits (123 bytes). Therefore, each transmitter node sends standard-Ethernet frames with a transmission rate of 128*60*984 bps (7,557,120 bps). Note that the total standard-Ethernet traffic of all the transmitter nodes is 90.65844 Mbps (7,557,120 bps*12), which is less than an output port's Ethernet bandwidth (100 Mbps). The switch fabric latency of an Ethernet switch is the same as that of a *RuggedSwitch* product (5.2 μs) [Rug08]. The analytical result shows that the best-case and worst-case switch latencies are 15.04 and 123.28 μs, respectively. In our implemented simulation, we set all transmitter nodes to transmit standard-Ethernet frames simultaneously. The simulation results and analytical results are shown in Fig. 4.29.

4.3.2.3 Multiple switches: In this scenario, there is a network comprising 12 transmitter nodes, three Ethernet switches, and one receiver, as depicted in Fig. 4.28. All transmitter nodes have the same standard-Ethernet traffic pattern as the one in the previous scenario, IEC 61850-9-1 Traffic. Therefore, all transmitter nodes send a standard-Ethernet frame with a bit-length of 984 bits (123 bytes) and a transmission period of 1/(128*60) seconds. The switch fabric latency of an Ethernet switch is the same as that of a *RuggedSwitch* product (5.2 μs) [Rug08]. The analytical result shows that the best-case and worst-case switch latencies are 30.08 and 138.32 μs, respectively. In our simulation, we set all transmitter nodes to transmit standard-Ethernet frames simultaneously. The simulation results and analytical results are shown in Fig. 4.29.

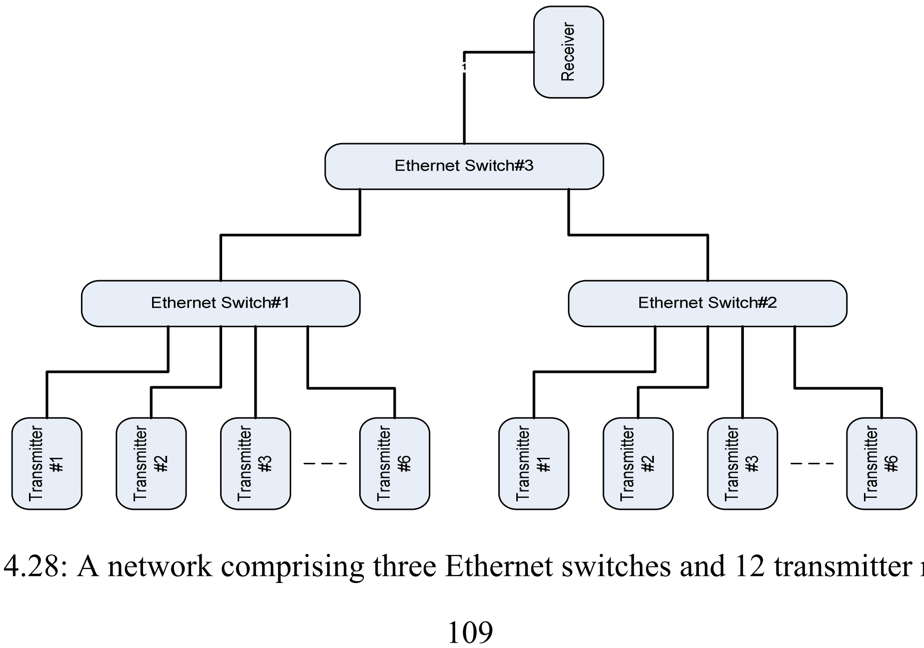
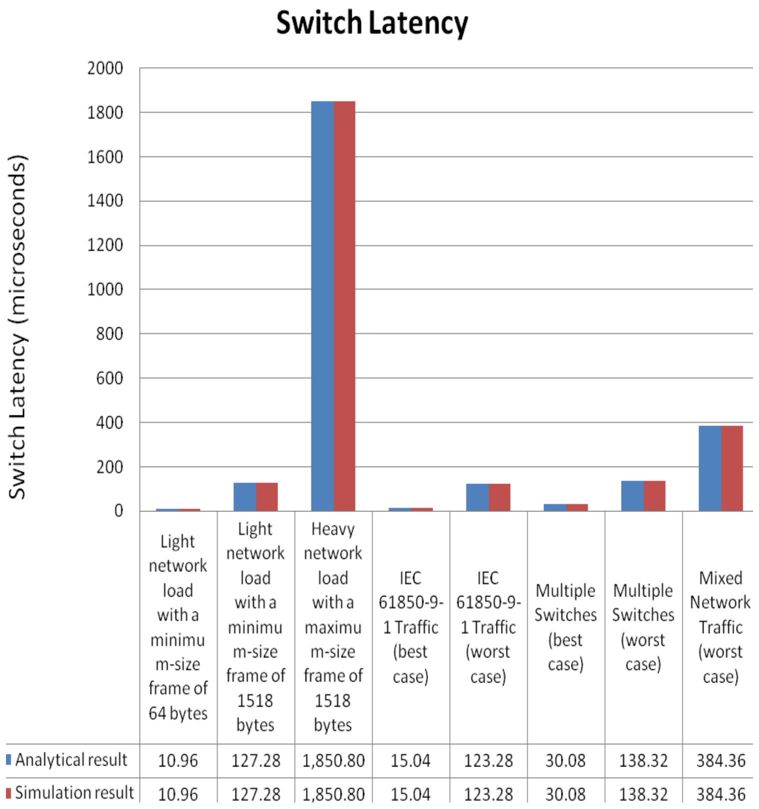


Fig. 4.28: A network comprising three Ethernet switches and 12 transmitter nodes

4.3.2.4 Mixed Network Traffic and Prioritization: In this scenario, the standard-Ethernet traffic is classified into high-priority and low-priority traffic. This network is structured the same as the network in the *Multiple switches* scenario. The Ethernet frames of sampled data (IEC 61850-9-1 Traffic) are defined as high-priority traffic, whereas the other standard-Ethernet frames are assigned as low-priority traffic. The switch fabric latency of an Ethernet-switch is the same as that of a *RuggedSwitch* product (5.2µs) [Rug08]. The analytical result shows that the worst-case switch latency of the high-priority standard-Ethernet frames is 384.36 µs. In our simulation model, we define high-priority standard-Ethernet frames as rate-constraint traffic (RC-traffic), and low-priority standard-Ethernet frames as best-effort traffic (BE-traffic). The low-priority standard-Ethernet frames are transmitted from all transmitter nodes with a transmission rate of 10/12 Mbps each. We set the total number of high-priority standard-Ethernet frames as 2,000 frames per transmitter node. The simulation results and analytical results are shown in Fig. 4.29.



	Light network load with a minimum-size frame of 64 bytes	Light network load with a minimum-size frame of 1518 bytes	Heavy network load with a maximum-size frame of 1518 bytes	IEC 61850-9-1 Traffic (best case)	IEC 61850-9-1 Traffic (worst case)	Multiple Switches (best case)	Multiple Switches (worst case)	Mixed Network Traffic (worst case)
🟦 Analytical result	10.96	127.28	1,850.80	15.04	123.28	30.08	138.32	384.36
🟥 Simulation result	10.96	127.28	1,850.80	15.04	123.28	30.08	138.32	384.36

Fig. 4.29: The results from the analytical model [Rug08] and our simulation

4.4 Summary

In this chapter, we have presented our simulation model for a TTE system. All components of the simulation model have been described. Due to the fact that the TT communication of a TTE system is predefined and deterministic, the TT communication in our implemented simulation can be validated with a predefined confliction-free communication schedule. Since the communication of standard Ethernet is non-deterministic, we have calibrated the simulation results from our simulation model with the analytical results from published papers [LL02] [Rug08]. The calibration results show that the results from the simulations based on our simulation model coincide with the analytical results.

CHAPTER 5

A TT-traffic Scheduling Approach for TTEthernet Systems

In this chapter we present our TT-traffic scheduling approach for TTEthernet systems. This approach aims to obtain all possible TT-communication schedules, and analyze the remaining bandwidth and timing property for standard-Ethernet traffic in TTEthernet systems. In TTEthernet systems, the standard-Ethernet traffic is used for non-real-time applications, whereas the TT traffic is used for real-time applications (including safety-critical ones). TT messages, which are transported with deterministic delays and low jitter, take precedence over standard-Ethernet frames. This denotes that the transport of standard-Ethernet frames cannot interfere with the timely behavior of TT-message communication. Since TT-message communication follows a TT-communication schedule, the performance of standard-Ethernet message communication depends not only on the amount of standard-Ethernet traffic, but also the TT-communication schedule (periods and offsets). In this chapter, we describe how to enumerate all the possible TT-communication schedules (based on our TT-traffic scheduling approach) from the specified periods of given time-critical applications. With our approach, an amount of bandwidth for standard-Ethernet traffic in a TTEthernet network is analyzed. We present two possible offset forms: *continuous offset form* and *distributed offset form*. Examples of time-critical applications are also given to illustrate our TT-traffic scheduling approach.

5.1 TTE Cluster cycle

A TTE-cluster cycle is a real-time cluster cycle for a TTE cluster. We refer to a TTE cluster cycle as simply a *cluster cycle* hereafter. All TT-message transmission events in a TTE cluster are scheduled in a cluster cycle. An event pattern of TT-message transmission is repeated every cluster cycle. The length of a cluster cycle is represented by a positive integer, approximately equal to the smallest period from all scheduled periods in a cluster cycle (as depicted in Fig. 5.1). Fig. 5.1 shows three time-critical applications (A, B, and C) scheduled in a cluster cycle. The applications have periods (2, 4, and 6 milliseconds, respectively). We can calculate the length of a cluster cycle by means of **L**east **C**ommon **M**ultiple (LCM). The LCM of 2, 4, and 6, denoted as LCM (2, 4, 6), is 12.

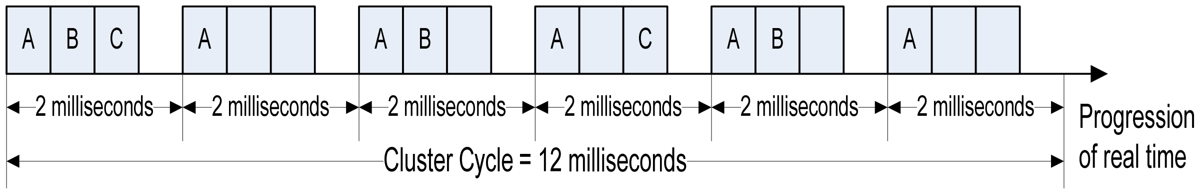


Fig. 5.1: A cluster cycle of three time-critical applications (A, B, C with periods of 2, 4, 6 msec.)

5.2 Period of TT traffic

The TT traffic of a time-critical application is scheduled in a cluster cycle using two parameters: *period* and *offset*. In this section we describe how to obtain all possible schedules from the periods of given time-critical applications. In our approach, we mathematically analyze an amount of remaining bandwidth for standard-Ethernet traffic. The schedule with the maximum remaining bandwidth for standard-Ethernet traffic is supposed to be the optimal one. We call this approach, which obtain that optimal schedule, the *remaining-bandwidth maximization* approach. This approach is illustrated with four time-critical applications. These time-critical applications (*appl_1, appl_2, appl_3,* and *appl_4*) have periods of 3, 4, 4, and 5 milliseconds, respectively. The Ethernet payload-data sizes of TT messages for the applications are 100, 46, 500, and 1000 bytes, respectively. Note that the Ethernet payload-data size of appl_2 is equal to that of a Protocol Control Frame (PCF). The *remaining-bandwidth maximization* approach is described as follows.

5.2.1 Enumerate all possible schedules from given periods

We define each *period* of a given time-critical application as a *based period.* In our example, the *based periods* of the four applications are *3, 4, 4,* and *5 milliseconds*. With three based periods in our example (the based periods of two applications are identical), we obtain three possible schedules for the four time-critical applications. We do this using the following two processes: *find satisfied based-periods of given time-critical applications*, and *calculate satisfied harmonic-periods of given time-critical applications.*

➢ *Find satisfied based-periods of given time-critical applications:* We can obtain satisfied based-periods of given time-critical applications as follows. Each based period of given time-critical applications is to be checked with the following satisfied condition: *if the based period of a time-critical application is satisfied, then it is equal or less than the lowest period of all given time-critical applications*. If the based period of a time-critical application is not satisfied (i.e. the lowest period of all the given time-critical applications is greater than the based period), then the based period is changed to half of its current value. The updated based period is then checked with the satisfied condition again. This process is repeatedly performed until the satisfied condition is reached. We call a based period that reaches the satisfied condition a *satisfied based-period*. In our example, the *satisfied based-periods* of the four applications are *3, 2, 2, and 2.5 milliseconds*, respectively. Note that the satisfied based-periods of two applications are identical (2 milliseconds). We conclude that the satisfied based-periods of the four time-critical applications are 2, 2.5, and 3 milliseconds.

- *Calculate satisfied harmonic-periods of given time-critical applications:* We define *harmonic periods* of a satisfied based-period as the number of satisfied based-periods or N*(satisfied based-periods), where N is a positive integer. *The satisfied harmonic-period of a time-critical application is the maximum of the time-critical application's harmonic periods,that is equal or less than the period of the time-critical application*. In our example, the satisfied harmonic-periods of the four time-critical applications (based on their respective *satisfied based-periods*) can be obtained as follows.
  - *Based on a satisfied based-period of 2 milliseconds:* The harmonic periods based on 2 milliseconds are {2, 4, 6, 8, 10, ... }. The satisfied harmonic-periods of the four applications are 2, 4, 4, and 4 milliseconds.
  - *Based on a satisfied based-period of 2.5 milliseconds*: The harmonic periods based on 2.5 milliseconds are {2.5, 5, 7.5, 10, 12.5, ... }. The satisfied harmonic-periods of the four applications are 2.5, 2.5, 2.5, and 5 milliseconds.
  - *Based on a satisfied based-period of 3 milliseconds*: The harmonic periods based on 3 milliseconds are {3, 6, 9, 12, 15, ... }. The satisfied harmonic-periods of the four applications are all equal to 3 milliseconds.

In our example we obtain three possible schedules from the satisfied harmonic-periods of the time-critical applications, as summarized in Table 5.1.

Table 5.1: The satisfied harmonic-periods of given time-critical applications in all possible schedules

Schedule	Satisfied based-period	Satisfied harmonic-periods			
		appl_1 (3 msec.)	appl_2 (4 msec.)	appl_3 (4 msec.)	appl_4 (5 msec.)
1sd schedule	2 msec.	2 msec.	4 msec.	4 msec.	4 msec.
2nd schedule	2.5 msec.	2.5 msec.	2.5 msec.	2.5 msec.	5 msec.
3rd schedule	3 msec.	3 msec.	3 msec.	3 msec.	3 msec.

5.2.2 Calculate a used bandwidth for TT traffic (UBTT)

After we obtain the satisfied harmonic-period of each schedule, we calculate a **U**sed **B**andwidth for **TT** traffic (UBTT) in each schedule, using Equation (5.1).

$$\mathrm{UBTT} = \sum_{i=1}^{n} \left(\frac{1}{\mathrm{period_i}} \times (\mathrm{FS_i} + \mathrm{IFG}) \right) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5.1)$$

Where, $\mathrm{period_i}$ is the satisfied harmonic-period of the ith time-critical application in a schedule,

$\mathrm{FS_i}$ is the total size of a frame in bits from the ith time-critical application,

IFG is the time interval of an inter-frame gap (96 bit-times).

Note that $\mathrm{FS_i}$ is a summation of Ethernet payload data, Ethernet overhead (14 bytes), and Frame Check Sequence (4 bytes). From the above example, we can calculate the UBTT for each of the obtained schedules as follows:

- UBTT of the first schedule: The satisfied harmonic-periods of appl_1, appl_2, appl_3 and appl_4 are 2, 4, 4, and 4 milliseconds. The Ethernet payload-data size of a TT-message for 100, 46, 500, and1000 bytes, respectively. The UBTTs of these applications are denoted as $\mathrm{UBTT_{appl\_1}}$, $\mathrm{UBTT_{appl2}}$, $\mathrm{UBTT_{appl3}}$ and $\mathrm{UBTT_{appl4}}$, respectively.

  $\mathrm{UBTT_{appl\_1}}$ = (1/(2*10^{-3}))*(100*8 + 14*8 + 4*8 + 96) = 520 kbps

  $\mathrm{UBTT_{appl\_2}}$ = (1/(4*10^{-3}))*(46*8 + 14*8 + 4*8 + 96) = 152 kbps

  $\mathrm{UBTT_{appl\_3}}$ = (1/(4*10^{-3}))*(500*8 + 14*8 + 4*8 + 96) = 1,060 kbps

  $\mathrm{UBTT_{appl\_4}}$ = (1/(4*10^{-3}))*(1000*8 + 14*8 + 4*8 + 96) = 2,060 kbps

  UBTT = (520 + 152 + 1,060 + 2,060) kbps= 3,792 kbps

- UBTT of the second schedule: The satisfied harmonic-periods of appl_1, appl_2, appl_3 and appl_4 are 2.5, 2.5, 2.5, and 5 milliseconds. The Ethernet payload-data size of a TT-message for these applications is 100, 46, 500, and 1000 bytes, respectively. The UBTTs of these applications are denoted as $\mathrm{UBTT_{appl\_1}}$, $\mathrm{UBTT_{appl2}}$, $\mathrm{UBTT_{appl3}}$ and $\mathrm{UBTT_{appl4}}$, respectively.

  $\mathrm{UBTT_{appl\_1}}$ = (1/(2.5*10^{-3}))*(100*8 + 14*8 + 4*8 + 96) = 416 kbps

  $\mathrm{UBTT_{appl\_2}}$ = (1/(2.5*10^{-3}))*(46*8 + 14*8 + 4*8 + 96) = 243.2 kbps

  $\mathrm{UBTT_{appl\_3}}$ = (1/(2.5*10^{-3}))*(500*8 + 14*8 + 4*8 + 96) = 1,696 kbps

  $\mathrm{UBTT_{appl\_4}}$ = (1/(5*10^{-3}))*(1000*8 + 14*8 + 4*8 + 96) = 1,648 kbps

  UBTT = (416 + 243.2 + 1,696 + 1,648) kbps = 4,003.2 kbps

- UBTT of the third schedule: The satisfied harmonic-periods of appl_1, appl_2, appl_3 and appl_4 are all 3 milliseconds. The Ethernet payload-data size of a TT-message for these applications is 100, 46, 500, and 1000 bytes, respectively. The UBTTs of these applications are denoted as $UBTT_{appl\_1}$, $UBTT_{appl2}$, $UBTT_{appl3}$ and $UBTT_{appl4}$, respectively.

  $UBTT_{appl\_1}$ = (1/(3*10-3)) * (100*8 + 14*8 + 4*8 + 96) = 346.67 kbps

  $UBTT_{appl\_2}$ = (1/(3*10-3)) * (46*8 + 14*8 + 4*8 + 96) = 202.67 kbps

  $UBTT_{appl\_3}$ = (1/(3*10-3)) * (500*8 + 14*8 + 4*8 + 96) = 1,413.33 kbps

  $UBTT_{appl\_4}$ = (1/(3*10-3)) * (1000*8 + 14*8 + 4*8 + 96) = 2,746.67 kbps

  UBTT = (346.67 + 202.67 + 1,413.33 + 2,746.67) kbps = 4,709.34 kbps

5.2.3 Calculate the remaining bandwidth for standard-Ethernet traffic (RBSE)

In a TTE system, standard-Ethernet frames can be transported through a TTE switch while no TT-messages in that switch are transmitting. The Ethernet bandwidth remaining after TT-traffic transmission is used for the transmission of standard-Ethernet traffic. We call this bandwidth used for standard-Ethernet frame transmission the *remaining bandwidth for standard-Ethernet traffic* (RBSE). In our example, we use Fast Ethernet, which has a bandwidth of 100 Mbps. The RBSE in a TTE network can be calculated using Equation 5.2. The calculated result of all the obtained schedules from the above example is shown below, where the first schedule gives the highest RBSE. We therefore conclude that the first schedule is the optimal schedule for the four time-critical applications. The optimal schedule consists of a set of periods for the four time-critical applications. These periods are 2, 4, 4, and 4 milliseconds. In the next section we consider the offsets of these periods.

RBSE = Network Bandwidth - UBTT, ……….…………..……………..……………(5.2)

, where

RBSE is the remaining bandwidth for standard-Ethernet traffic,

The Fast-Ethernet's bandwidth is 100 Mbps, UBTT is the used bandwidth for TT traffic.

In our example, we calculate the RBSE as follows:

- First schedule: RBSE = 100 Mbps - 3,792 kbps = **96,208 kbps**
- Second schedule: RBSE = 100 Mbps - 4,003.2 kbps = 95,996.8 kbps
- Third schedule: RBSE = 100 Mbps - 4,709.34 kbps = 95,290.66 kbps

5.3 Offset of TT traffic

In this section we describe how to obtain the offsets of time-critical applications for a schedule. In our TT-traffic scheduling approach, the schedule we use to find the offsets is a result of the previous section *Period of TT traffic* (i.e. the schedule is an optimal schedule from the previous section). We present two possible offset forms: *continuous offset form* and *distributed offset form*. In the *continuous offset form*, TT messages from time-critical applications are transmitted contiguously, as depicted in Fig. 5.2(a). In the *distributed offset from*, standard-Ethernet frames are able to be transmitted between any two consecutive TT messages, as depicted in Fig. 5.2(b). In TTEthernet systems, standard-Ethernet frames cannot be transmitted at the same time as a TT message, as depicted in Fig. 5.3. Once a standard-Ethernet frame arrives at the transmitter of an Ethernet controller, a TTEthernet engine checks whether the transmission of that standard-Ethernet frame will fall in a TT-message transmission interval or not. If the TTEthernet engine finds that the transmission of a standard-Ethernet frame will fall in a TT-message transmission interval, the standard-Ethernet frame has to wait until the TT-message transmission has completed. This implies that the transport delay of a standard-Ethernet frame depends not only on the Ethernet frame queuing in Ethernet switches, but also on the TT-message transmission schedule. In Fig. 5.3, it is shown that regardless of Ethernet-frame queuing, a standard-Ethernet frame that arrives at the transmitter of an Ethernet controller has to wait (at most) for the TT-message transmission interval plus the transmission time of the standard-Ethernet frame and an inter-frame gap (IFG). In the distributed offset form, a TT-message transmission interval is reduced in comparison to the interval in continuous offset form. This implies that the worst-case transport delay of a standard-Ethernet frame with a TT-communication schedule using the distributed offset form is smaller than the one in the same TTE network with a TT communication schedule using the continuous offset form. We describe both offset forms in detail in section 5.3.1 and 5.3.2.

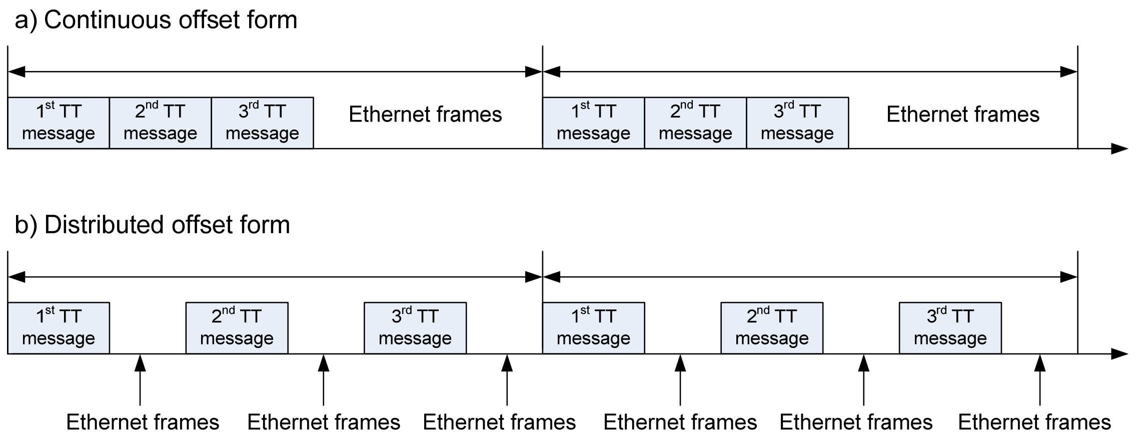


Fig. 5.2: Ethernet and TT message transmission in continuous offset form (a), and distributed offset form (b)

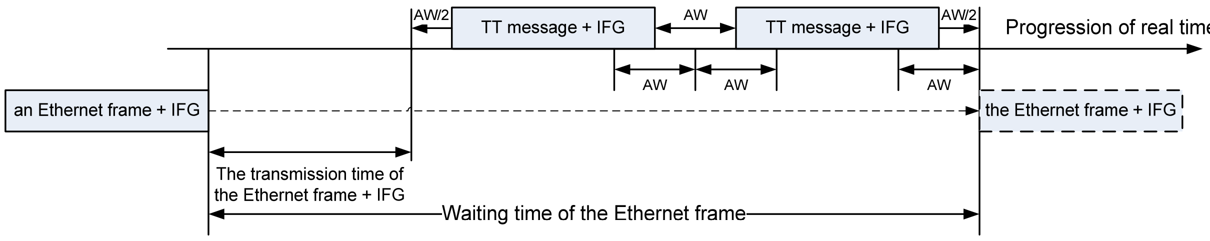


Fig. 5.3: Timing diagram of the transmission of two TT messages and one Ethernet frame

5.3.1 Continuous offset form

In principle of continuous offset form, TT messages are transmitted contiguously, as depicted in Fig. 5.4. This means that no standard-Ethernet frames can be transmitted in between contiguous TT messages. Standard-Ethernet frames are therefore transmitted in the remaining time, after the transmission of a set of contiguous TT messages has finished.

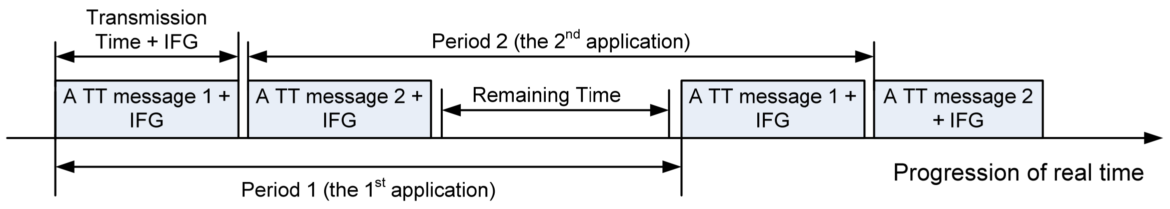


Fig. 5.4: The TT-traffic transmission of two time-critical applications in continuous offset form

5.3.1.1 A time interval between two contiguous messages in continuous offset form

In this section we analyze a time interval between two contiguous messages (i.e. between two contiguous TT-messages and between a TT message and a PCF) in a TTEthernet system. In a TTEthernet network, when all TTE devices are synchronized with each other, a TT message or a PCF is transported through the TTEthernet network without waiting for any other messages in any TTE device. This is because a TT-communication schedule in a TTEthernet system is defined to be confliction-free. In order to assign offsets in a confliction-free TT-communication schedule, a time interval between two consecutive messages is analyzed. In this analysis, we assume that a TT message is forwarded by a TTE switch in multicast form (i.e. a TT message received at a TTE switch is duplicated and then forwarded to all other ports of the TTE switch). According to the TTEthernet specification [TTE08], the offset of PCF traffic in a TT communication schedule is zero ($\text{Offset}_{\text{PCF}} = 0$). TT messages can be transmitted only after all TTE devices have completely received PCFs. A PCF starts traveling from a synchronization master to a compression master in the same TTE cluster. The compression master compresses the received PCF and then distributes it to all synchronization masters and synchronization clients. The time it takes for a PCF to be transmitted from a synchronization master to a compression master and back to a node (TD_{PCF}) is shown in Equation 5.3. The compression master delay of TTEthernet is the time it takes for a compression master to relay a PCF, and is described in detail in the TTEthernet specification [TTE08]. The time interval between a PCF and a TT message is equal to TD_{PCF}, as depicted in Fig. 5.5.

$$TD_{PCF} = 2 * D_{MTD} + D_{CMD} + D_{CCD} + IFG + D_{AW}, \qquad (5.3)$$

, where TD_{PCF} is a PCF transport delay from a synchronization master to a compression master and back to a node (in the same TTE cluster), D_{MTD} is the maximum transmission delay of TTEthernet, D_{CMD} is the compression master delay of TTEthernet, D_{CCD} is the clock correction delay of TTEthernet, IFG is a time interval of an inter-frame gap, and D_{AW} is the acceptance window of TTEthernet.

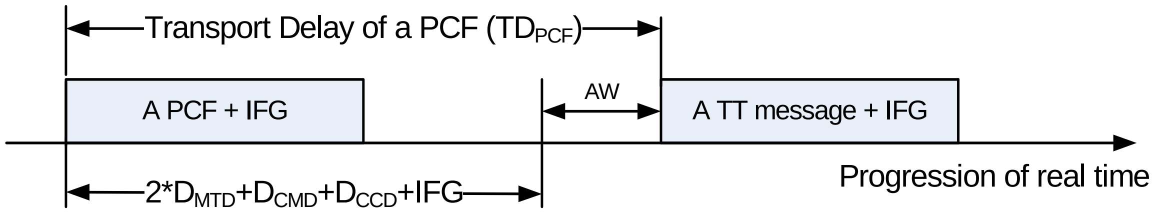

Transport Delay of a PCF (TD$_{PCF}$)
A PCF + IFG
AW
A TT message + IFG
Progression of real time
2*D$_{MTD}$+D$_{CMD}$+D$_{CCD}$+IFG

Fig. 5.5: A time interval between a PCF and a TT message

1. With regard to the transmission of two contiguous TT-messages, a time interval between them in continuous offset form depends on their sender node(s). Therefore, there are two scenarios of TT-message transmission in continuous offset form with regard to sender nodes. These are described as follows: *Two contiguous TT-messages sent from the same sender node*: The second TT-message of these two contiguous TT-messages can be transmitted when the transmission of the first TT-message at the same sender node is completed (Fig. 5.6). The time interval between two TT-messages is equal to an inter-frame gap plus the acceptance window of TTEthernet. The time interval between two consecutive TT-message's dispatch points in time is derived from Equation 5.4.

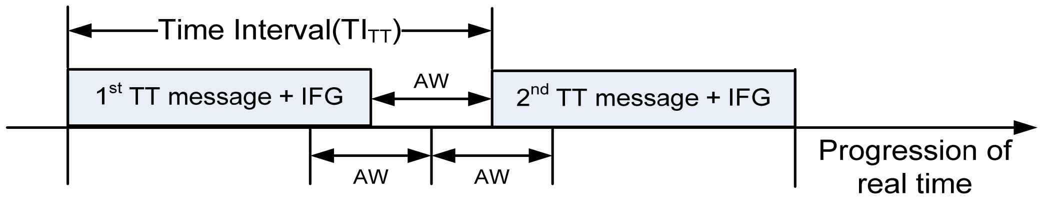


Fig. 5.6: Two contiguous TT-messages from the same sender node

in continuous offset form

$$TI_{TT} = TT_{TT\_1} + D_{IFG} + D_{AW} \text{ , } \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5.4)$$

, where TI$_{TT}$ is the time interval between two contiguous TT-message's dispatch points in time, TT$_{TT\_1}$ is the transmission time of the first TT message of two contiguous TT-messages, D$_{IFG}$ is a time interval of an inter-frame gap, and D$_{AW}$ is the acceptance window of TTEthernet.

2. *Two contiguous TT-messages sent from two different sender nodes*: We call a node transmitting the first and second TT-message of two contiguous TT-messages the *first node* and *second node*, respectively. We assume that TT messages are transmitted in a multicast form. Thus, the second TT-message can be transmitted from the second node after the first TT-message from the first node has been completely transmitted to the second node. The time interval between the two contiguous TT-messages depends on TTE-network delays between both nodes, as shown in Fig. 5.7. The time interval between the two contiguous TT-message's dispatch points in time is derived from Equation 5.5. Note that the time interval between two consecutive TT-message's dispatch points in time (from different sender nodes) is equal to the one from the sender plus the TTE-network delay between both nodes.

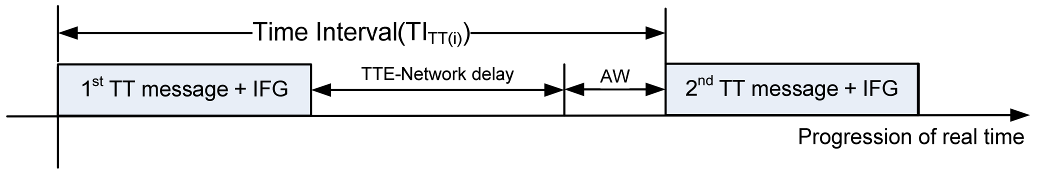


Fig. 5.7: Two contiguous TT-messages from two different sender nodes

in continuous offset form

$$TI_{TT(i)} = TT_{msg(i)} + D_{IFG(i)} + D_{AW} + ND_{TTE(i)} \text{ , } \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5.5)$$

$$ND_{TTE(i)} = DD_{sender(i)} + PD_{sender(i)} + \sum_{j=1}^{N_{sw}} LSW_{(i,j)} \text{ ,} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5.6)$$

$$LSW_{(i,j)} = TT_{msg(i,j)} + D_{IFG(i,j)} + PD_{(i,j)} + DD_{(j)} + TTL_{(j)} \text{ ,} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5.7)$$

, where

$TI_{TT(i)}$ is the time interval between the i^{th} and $(i+1)^{th}$ TT-message's dispatch points in time,

$TT_{msg(i)}$ is the transmission time of the i^{th} TT message,

D_{IFG} is a time interval of an inter-frame gap,

D_{AW} is the acceptance window of TTEthernet,

$ND_{TTE(i)}$ is the transport delay of the i^{th} TT message from the i^{th} TT message's sender node to the $(i+1)^{th}$ TT message's sender node.

$DD_{sender(i)}$ is the dispatch delay of the i^{th} TT message at the i^{th} TT message's sender node,

$PD_{sender(i)}$ is the propagation delay of an Ethernet network cable connected to the i^{th} TT message's sender node,

$LSW_{(i,j)}$ is the latency of the j^{th} TTE switch for relaying the i^{th} TT message,

N_{SW} is the total number of TTE switches which the i^{th} TT messages passes through between the i^{th} TT message's sender node to the $(i+1)^{th}$ TT messages's sender node,

$TT_{msg(i,j)}$ is the transmission time of the i^{th} TT message at the j^{th} TTE switch,

$D_{IFG(i,j)}$ is the time interval of an inter-frame gap of the i^{th} TT message at the j^{th} TTE switch,

$PD_{(i,j)}$ is the propagation delay of an Ethernet network cable where the i^{th} TT message is transmitted out at the j^{th} TTE switch,

DD_j is the dispatch delay of the j^{th} TTE switch for a TT message,

TTL_j is the TT latency of the j^{th} TTE switch for relaying a TT message.

5.3.1.2 A remaining time in a satisfied based-period in continuous offset form

In this section we analyze a time interval not used for TT-traffic transmission within a satisfied based-period in continuous offset form (depicted in Fig. 5.8). We call this time interval *a remaining time of a satisfied based-period*. The remaining time of a satisfied based-period is a time interval that remains from the TT-traffic transmission of time-critical applications, which are already scheduled within the satisfied based-period. Other TT traffic can be scheduled in the remaining time of a satisfied based-period. In addition, a TTE device can utilize the remaining time of a satisfied based-period to transport standard-Ethernet frames, if the remaining time is sufficient to transport them without any interference to the timely behavior of TT-traffic. Fig. 5.8(b) depicts the transmission of two TT messages within a satisfied based-period in continuous offset form. The remaining time of a satisfied based-period can be calculated using Equation 5.8.

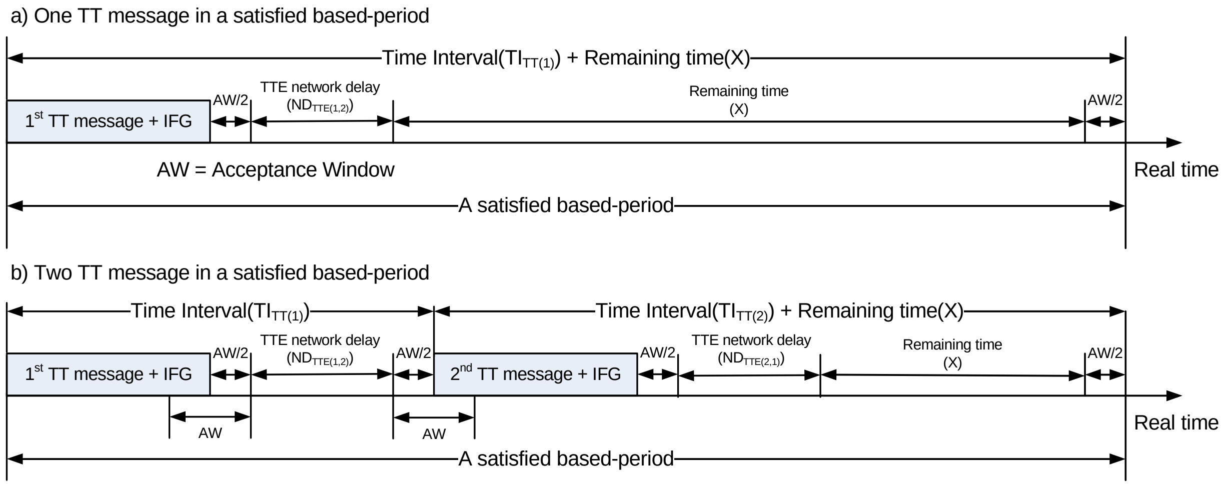

a) One TT message in a satisfied based-period
Time Interval($TI_{TT(1)}$) + Remaining time(X)
1st TT message + IFG
AW/2
TTE network delay ($ND_{TTE(1,2)}$)
Remaining time (X)
AW/2
AW = Acceptance Window
Real time
A satisfied based-period
b) Two TT message in a satisfied based-period
Time Interval($TI_{TT(1)}$)
Time Interval($TI_{TT(2)}$) + Remaining time(X)
1st TT message + IFG
AW/2
TTE network delay ($ND_{TTE(1,2)}$)
AW/2
2nd TT message + IFG
AW/2
TTE network delay ($ND_{TTE(2,1)}$)
Remaining time (X)
AW/2
AW
AW
Real time
A satisfied based-period

Fig. 5.8: The remaining time of a satisfied based-period in continuous offset form

$$X = SBP - \sum_{i=1}^{N} TI_{TT(i)} , \ldots\ldots (5.8)$$

, where

X is the remaining time of a satisfied based-period,

SBP is the satisfied based-period of a TT-communication schedule,

$TI_{TT(i)}$ is the time interval between the ith and (i+1)th TT-message's dispatch points in time,

N is the number of TT messages in the satisfied based-period.

5.3.1.3 Offsets in continuous offset form

In a TT communication schedule, the offset of PCF traffic is always zero. In Fig. 5.9, the first offset of TT traffic ($Offset_{TT\_1}$) is equal to the time it takes for a PCF to be transmitted from a synchronization master to a compression master and then back to a node (TD_{PCF}), as derived from Equation 5.3. The second offset of TT traffic ($Offset_{TT\_2}$) is equal to the first offset of TT traffic ($Offset_{TT\_(1)}$) plus the time interval between the two contiguous TT message's dispatch points in time (TI_{TT}), as derived from Equation 5.4 or Equation 5.5. All offsets of TT traffic in a TTE cluster can be calculated using Equation 5.9.

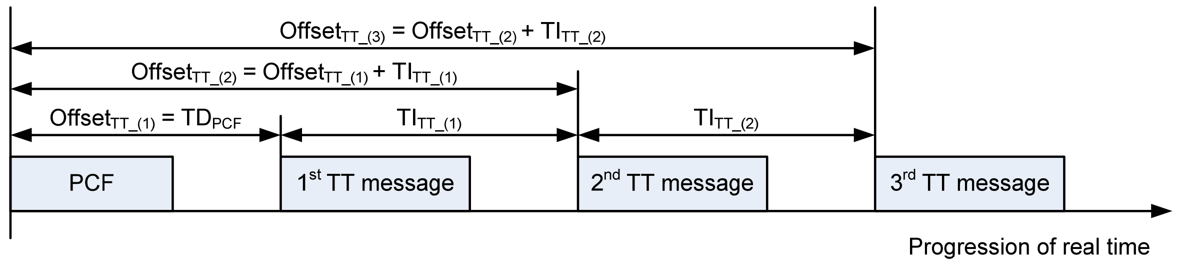

$Offset_{TT\_(3)} = Offset_{TT\_(2)} + TI_{TT\_(2)}$
$Offset_{TT\_(2)} = Offset_{TT\_(1)} + TI_{TT\_(1)}$
$Offset_{TT\_(1)} = TD_{PCF}$
$TI_{TT\_(1)}$
$TI_{TT\_(2)}$
PCF
1st TT message
2nd TT message
3rd TT message
Progression of real time

Fig. 5.9: The offsets of a PCF and three TT messages in a TT-communication schedule

$$Offset_{TT\_(j)} = TD_{PCF} + \sum_{i=1}^{j-1} TI_{TT\_(i-1)} \quad ,\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5.9)$$

, where

$Offset_{TT\_(j)}$ is the offset of TT traffic of the jth time-critical application,

TD_{PCF} is the PCF transport delay from a synchronization master to a compression master and back to any node in a TTE cluster, as derived from Equation 5.3,

$TI_{TT\_(i-1)}$ is the time interval between the (i-1)th TT message and the ith TT message, as derived from Equation 5.4 and 5.5. Note that $TI_{(0)}$ is equal to zero.

5.3.1.4 Continuous offset form in a TTE cluster cycle

In the previous sections we have introduced the fundamental concept of continuous offset form. In this section we extend this concept into a TTE-cluster cycle. Based on continuous offset form, we can obtain the offsets of time-critical applications in a TTE-cluster cycle, by using the method described below. In this method, one of the given time-critical applications takes responsibility to generate PCFs (synchronization messages). Our method is illustrated with four time-critical applications. These applications are called *appl_1*, *appl_2*, *appl_3* and *appl_4*. They have periods of 2, 4, 4, and 6 milliseconds, respectively. The Ethernet-payload-data sizes of a TT message for the four applications are 100, 46, 500, and 1000 bytes, respectively.

1) *Sort the given time-critical applications:* We sort the given time-critical applications in ascending order with regard to their periods. The first time-critical application with the lowest period (post sorting) has the highest scheduling priority in a TTE-cluster cycle. Note that the time-critical application with the lowest period will transmit a TT message in every

time interval of the satisfied based-period along a TTE-cluster cycle. In our example, the sorted order of the time-critical applications is: appl_1, appl_2, appl_3 and appl_4. The sorted time-critical applications will be scheduled in ascending order for the next step (i.e. appl_1 is the time-critical application to be considered first in the next step).

2) *Calculate the number of time-slots for a TTE-cluster cycle and time-critical applications:* The lowest period of time-critical applications in a schedule is the satisfied based-period of the schedule. A TTE-cluster cycle is equal to the Least Common Multiple (LCM) of all time-critical application periods. The TTE-cluster cycle also denotes a number of satisfied based-periods. We call a satisfied based-period a *time-slot*. A TTE cluster cycle consists of a set of the LCM number of connected time-slots. We call all connected time-slots in a TTE-cluster cycle *cluster-cycle time-slots,* as depicted in Fig. 5.10. The number of cluster-cycle time-slots is denoted as $NTS_{cluster}$. In our example, the lengths of the TTE-cluster cycle and the time slot are equal to 12, and 2 milliseconds, respectively. Therefore, the number of cluster-cycle time-slots ($NTS_{cluster}$) is 6. We also define the number of time-slots for one period of a time-critical application (NTS_{appl}) in a schedule as the satisfied harmonic-period of the time-critical application divided by the satisfied based-period. We denote the number of time-slots for one period of a time-critical application as NTS_{appl}. The number of time-slots for appl_1's period is one(NTS_{appl_1}). The number of time-slots for appl_2's period is two (NTS_{appl_2}). The number of time-slots for appl_3's period is two (NTS_{appl_3}). The number of time-slots for appl_4's period is three time-slots (NTS_{appl_4}). This denotes that one TT message from appl_1 is transmitted every time-slot. One TT message from appl_2 and appl_3 are transmitted every two time-slots. One TT message from appl_4 is transmitted every three time-slots.

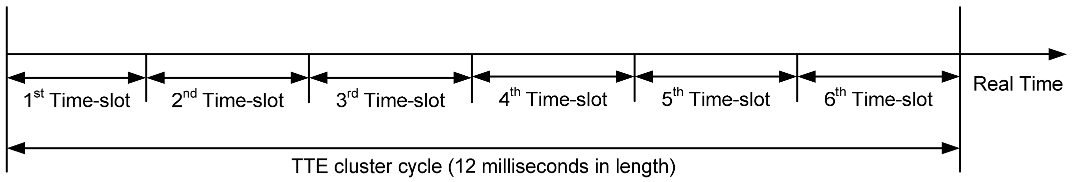


Fig. 5.10: The cluster-cycle time-slots in a TTE cluster cycle

3) *Put PCFs and TT messages from the sorted time-critical applications into time-slots*: In this process we consider each of the sorted time-critical applications in ascending order. The first time-critical application (post sorting) is the first to put its messages into the time-slots. We put PCFs and TT messages from each of the sorted time-critical applications into time-slots using the following steps:

<u>Step 1</u>: Find possible sets of time-slots for scheduling TT-messages from a time-critical application: The number of possible sets of time-slots is equal to the number of time-slots for a time-critical application (NTS_{appl}). Due to the periodic transmission of TT traffic, a set of time-slots for scheduling TT-messages is defined using Equation 5.10. In our example,

appl_1 (NTS_{appl_1}) has one time-slot. Thus, appl_1 has the following set of time-slots: {1st, 2nd, 3rd, 4th, 5th, 6th}. For appl_2, NTS_{appl_2} is two time-slots. Thus there are two set of time-slots, which are {1st, 3rd, 5th} and {2nd, 4th, 6th}. For appl_3, there are also two sets of time-slots: {1st, 3rd, 5th} and {2nd, 4th, 6th}. For appl_4 there are three set of time-slots: {1st, 4th}, {2nd, 5th} and {3rd, 6th}.

$TSn = TS1 + (n-1)*NTS_{appl}$, $TSn \leq NTS_{cluster}$,……………………………………………..(5.10)

, where TSn is the set of time-slots for a time-critical application {TS1, TS2, …, TSn},

TS1 is the firstly-selected time-slot for a time-critical application,

n is a positive integer = {1,2,3,…},

NTS_{appl} is the number of time-slots for a time-critical application,

$NTS_{cluster}$ is the total number of time-slots in a TTE cluster cycle.

Step 2: Find the lowest remaining time of each set of time-slots: The remaining time of each time-slot in a set of time-slots is calculated using Equation 5.8. The lowest remaining time in a set of time-slots is used to put a TT message or PCF in all time-slots. For instance, the remaining time of the 3rd time-slot is lowest in the first set of time-slots of appl_3 {the 1st, 3rd and 5th time-slots}. Thus, the TT-message's dispatch point in time for appl_3 is the time interval of a time-slot minus the lowest remaining time. This dispatch point in time in the 3rd time-slot is used for the dispatch points in time in other time-slots {the 3rd and 5th time-slots} in the first set of time-slots {the 1st, 3rd, and 5th time-slots}, as shown in Fig. 5.11(c). Note that a dispatch point in time in a time-slot is not greater than the time interval of the time-slot. Before putting a TT message in a set of time-slots, the lowest remaining time of a set of time-slots is checked to see whether or not it is sufficient to schedule the TT message. This is described in the following step.

Step 3: Check the lowest remaining time of each set of time-slots with the sufficient condition: The lowest remaining time of a set of time-slots for a time-critical application will be checked with the sufficient condition. *The sufficient condition is that the remaining time of a time-slot is equal to or greater than zero, after putting a TT message or PCF from a time-critical application into the time-slot.* If the lowest-remaining time-slot of a set of time-slots does not satisfy the sufficient condition, the lowest remaining time of the next set of time-slots (of the same time-critical application) will be checked. If no set of time-slots satisfies the sufficient condition, the process will stop and a scheduling failure is reported.

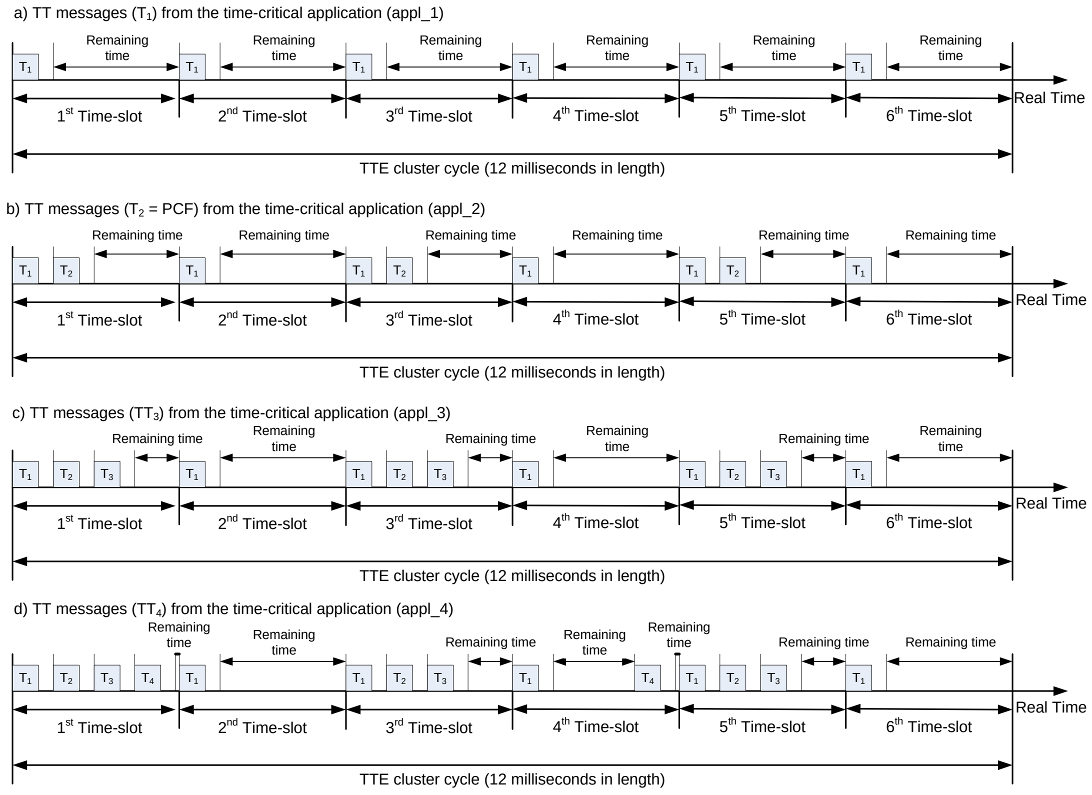

a) TT messages (T1) from the time-critical application (appl_1)
b) TT messages (T2 = PCF) from the time-critical application (appl_2)
c) TT messages (TT3) from the time-critical application (appl_3)
d) TT messages (TT4) from the time-critical application (appl_4)
Remaining time
T1
T2
T3
T4
Real Time
1st Time-slot
2nd Time-slot
3rd Time-slot
4th Time-slot
5th Time-slot
6th Time-slot
TTE cluster cycle (12 milliseconds in length)

Fig. 5.11: TT messages scheduled in time-slots in continuous offset form

In our example, appl_1 has one set of time-slots (the 1st, 2nd, 3rd, 4th, 5th and 6th time-slots). A TT message from appl_1 is put in each time-slot in this set of time-slots, as shown in Fig. 5.11(a). For appl_2, there are two sets of time-slots, the 1st, 3rd and 5th time-slots and the 2nd, 4th, and 6th time-slots. The first set of time-slots for appl_2 (the lowest remaining time in the set of time-slots) is initially checked with the sufficient condition. If the first set of time-slots for appl_2 satisfies the sufficient condition, a TT message from appl_2 will be put in each time-slot in the first set of time-slots, as shown in Fig. 5.11(b). For appl_3, there are also two sets of time-slots, the 1st, 3rd and 5th time-slots and the 2nd, 4th and 6th time-slots}. The first set of time-slots is considered first. If the first set of time-slots satisfies the sufficient condition, a TT message from appl_3 will be put in each time-slot in the first set of time-slots, as shown in Fig. 5.11(c). For appl_4, there are three sets of time-slots, the 1st and 4th time-slots, the 2nd and 5th time-slots and the 3rd and 6th time-slots. If the first set of time-slots satisfies the sufficient condition, a TT message from appl_4 will be put in each time-slot in the first set of time-slots, as shown in Fig. 5.11(d). In cases where the first set of time-slots does not satisfy the sufficient condition, the second set of time-slots will be checked. If the second set of time-slots for appl_4 does not satisfy the sufficient condition, the third set of time-slots will be checked. If no set of time-slots satisfies the sufficient condition, the process stops and a scheduling failure is reported.

4) *Find the offsets of PCF and TT traffic in a TTE-cluster cycle:* In this process we calculate the offsets of PCF and TT traffic from the result of the previous process. By using Equation 5.3, 5.4 and 5.5, a time interval between any two contiguous message's dispatch points in time (TI_{TT}) along a time-slot can be calculated. The offset of a message within a time-slot is the summation of the time intervals (TI_{TT}(s)) of the preceding messages within the message's time-slot, plus time intervals of all preceding time-slots. In our example, the offset of the first, second, third and fourth time-critical applications ($Offset_{TT(1)}$, $Offset_{TT(2)}$, $Offset_{TT(3)}$ and $Offset_{TT(4)}$, respectively) are shown below.

$Offset_{TT(1)} = 0$

$Offset_{TT(2)} = Offset_{PCF} = TD_{PCF}$, T_2 is a PCF

$Offset_{TT(3)} = TD_{PCF} + TI_{TT(2)}$

$Offset_{TT(4)} = TD_{PCF} + TI_{TT(2)} + TI_{TT(3)}$

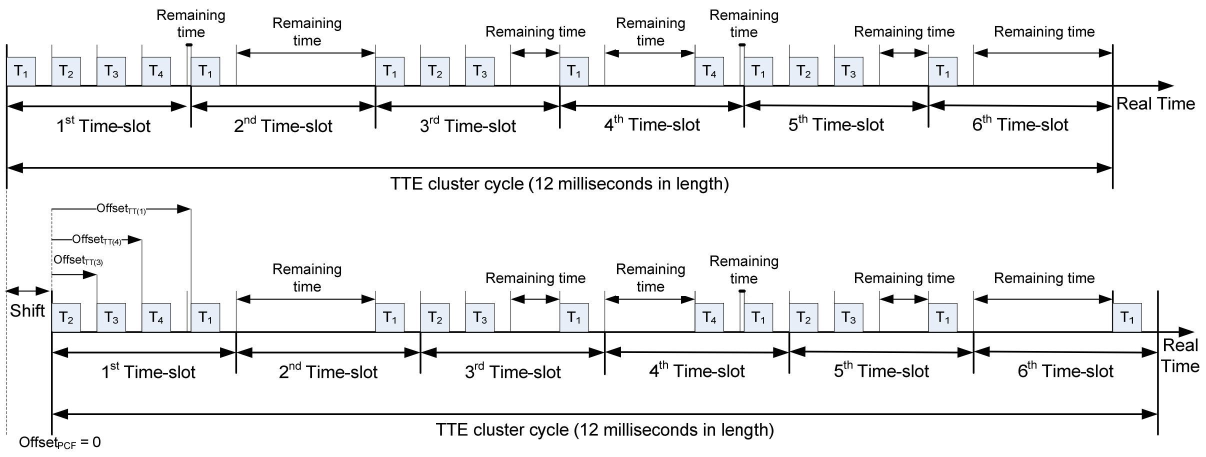


Fig. 5.12: The offsets of TT traffic before and after the shifting process (continuous offset form)

Since the offset of PCF traffic in TTEthernet is defined as zero, all offsets will be shifted with the offset of PCF traffic. This is, that all the offsets are to be subtracted from the PCF offset ($Offset_{PCF}$). If the result from the subtraction (the new offset) is negative, the new offset will be added with the period of the time-critical application which it belongs to. Thus, all new offsets are positive values. The result from shifting all the offsets with the PCF offset is shown below, and is depicted in Fig. 5.12.

$Offset_{TT(1)} = Period_{TT(1)} - TD_{PCF}$

$Offset_{TT(2)} = Offset_{PCF} = 0$

$Offset_{TT(3)} = TD_{PCF} + TI_{TT(2)} - TD_{PCF} = TI_{TT(2)}$

$\text{Offset}_{TT(4)} = \text{TD}_{PCF} + \text{TI}_{TT(2)} + \text{TI}_{TT(3)} - \text{TD}_{PCF} = \text{TI}_{TT(2)} + \text{TI}_{TT(3)}$

5) Schedule all TT messages and PCFs in a TTE-cluster cycle: In the previous processes we obtained all the offsets of time-critical applications. In this process, we calculate all dispatch instants of PCFs and TT messages along a TTE cluster cycle using Equations 5.11 and 5.12. PCFs arrive at a TTE device at the receive points in time. These are calculated using Equation 5.13 for a synchronization master or a synchronization client, or Equation 5.14 for a compression master. TT messages are received by a TTE device at receive points in time. These are calculated using Equation 5.15.

$$DIT_{PCF} = (n-1)(Period_{PCF}), n = \{1, 2, 3, \dots, \frac{Cluster\ cycle}{Period_{PCF}}\} \text{ ....(5.11)}$$

$$DIT_{TT(i)} = Offset_{TT(i)} + (n-1)\left(Period_{TT(i)}\right), n = \{1, 2, 3, \dots, \frac{Cluster\ cycle}{Period_{TT(i)}}\} \text{ .....(5.12)}$$

$$\text{RIT}_{PCF(SM/SC)} = \text{DIT}_{PCF} + \text{T}_{PCF} + \text{IFG} + TD_{PCF}, \text{ ....(5.13)}$$

$$\text{RIT}_{PCF(CM)} = \text{DIT}_{PCF} + \text{T}_{PCF} + \text{IFG} + MTD_{PCF}, \text{ ....(5.14)}$$

$$\text{RIT}_{TT(i,j)} = \text{DIT}_{TT(i)} + \text{T}_{TT(i)} + \text{IFG} + \text{ND}_{TTE(i,j)}, \text{ ....(5.15)}$$

, where DIT_{PCF} is the dispatch points in time for PCF traffic in a TTE cluster cycle,

Period_{PCF} is the transmission period of PCF traffic,

$\text{DIT}_{TT(i)}$ is the dispatch points in time for TT traffic of the ith time-critical application in a TTE cluster cycle,

$\text{Offset}_{TT(i)}$ is the offset of TT traffic of the ith time-critical application in a TTE cluster cycle.

T_{PCF} is the transmission time of a PCF = (PCF's length)/(Ethernet bandwidth),

IFG is the time interval of an inter-frame gap,

TD_{PCF} is the PCF transport delay from a synchronization master to a compression master and back to any node in the same TTE cluster (Equation 5.3),

$\text{RIT}_{PCF(SM/CM)}$ is the receiving points in time for PCF messages at a synchronization master or a synchronization client,

MTD_{PCF} is the maximum transmission delay of a TTE system,

$\text{RIT}_{PCF(SM/CM)}$ is the receiving points in time for PCF messages at a compression master,

$\text{RIT}_{TT(i,j)}$ is the receiving points in time for TT messages of the ith time-critical application at the jth TTE device,

$T_{TT(i)}$ is the transmission time of a TT message of the ith time-critical application (TT message's length)/(Ethernet bandwidth),

$ND_{TTE(i,j)}$ is the TTE-network delay of a TT message of the ith time-critical application from the TT message's sender node to the jth TTE device.

5.3.2 Distributed offset form

In principle of distributed offset form, standard-Ethernet frames can be transmitted between any two consecutive TT-messages, as depicted in Fig. 5.13. This means that a time interval between any two consecutive TT-messages can be used to transmit standard-Ethernet frames, if and only if the time interval is sufficient to transmit the standard-Ethernet frames. We assume that standard-Ethernet frames have an equal possibility of arriving at a TTE device at any point in time. Therefore, we equally distribute the amount of remaining time for standard-Ethernet frame transmission within a time interval of a satisfied based-period. Due to unfixed sizes of standard-Ethernet frames (46-1500 bytes of payload data), we define the remaining time for transmitting standard-Ethernet frames to be at least equal to a maximum-sized Ethernet frame, plus an inter-frame gap. With such a remaining time, at least one standard-Ethernet frame can be transmitted.

5.3.2.1 A remaining time in a satisfied based-period in distributed offset form

In this section we analyze a remaining time for standard-Ethernet frame transmission within a time interval of a satisfied based-period in distributed offset form. A TTE device can utilize this remaining time to transmit standard-Ethernet frames. We assume that standard-Ethernet frames have an equal possibility of arriving at a TTE device at any point in time. Therefore, an interval of a remaining time between any two TT messages within a satisfied based-period is the same as any other within that satisfied based-period, as depicted in Fig. 5.13 (X1 = X2). Additionally, a remaining time for standard-Ethernet transmission shall be equal or greater than the transmission time of a maximum-sized Ethernet frame, so that a maximum-sized Ethernet frame can be transmitted. In distributed offset form, a remaining time for standard-Ethernet frame transmission within a time interval of a satisfied based-period can be calculated using Equations 5.16, 5.17 and 5.18.

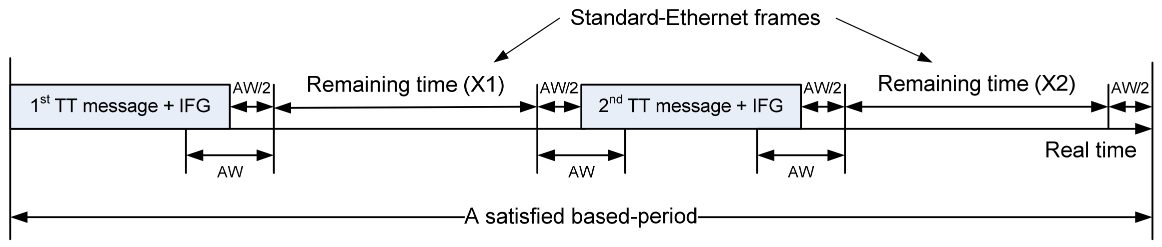


Fig. 5.13: The transmission of two TT messages in distributed offset form

$X = X_j * N = SBP - \{Tr_{TT(i)} + (AW + IFG) * n\}$, ……….…..……..……..………........(5.16)

$Tr_{TT(i)} = \frac{1}{BW}\sum_{i=0}^{n} msg\_size_i$, ……………………………….....……………….….(5.17)

$X = X_j * N = X_1 + X_2 + X_3 + \cdots + X_N = \sum_{j=1}^{N} Xj$,………...…..…………………..……...(5.18)

, where X is the overall remaining time for standard-Ethernet traffic in a satisfied based-period,

X_j is the jth remaining time for standard-Ethernet traffic within the satisfied based-period,

SBP is the satisfied based-period of a TT communication schedule,

msg_size_i is the size of the ith frame in bits within the satisfied based-period,

n is the number of TT messages in the satisfied based-period,

IFG is a time interval of an inter-frame gap,

BW is the Ethernet bandwidth,

AW is the acceptance window of TTEthernet,

N is the number of TT messages in the satisfied based-period,

$Tr_{TT(i)}$ is the transmission time of the ith TT-message.

In cases where *a remaining time between any two TT-messages (X_j) is less than the transmission time of a maximum-sized standard-Ethernet frame plus an inter-frame gap (IFG),* there shall be at least one remaining time that is equal or greater than the transmission time of a maximum-sized standard-Ethernet frame plus an IFG. This is so the problem of a maximum-sized standard-Ethernet frame not being able to pass through a TTE device can be avoided. In this case a remaining time between any two consecutive TT messages is equal to the transmission time of a maximum-sized standard-Ethernet frame. If a remaining time is less than the transmission time of a maximum-sized standard-Ethernet frame, TT messages within this remaining time will be scheduled in continuous offset form, as depicted in Fig. 5.14(c).

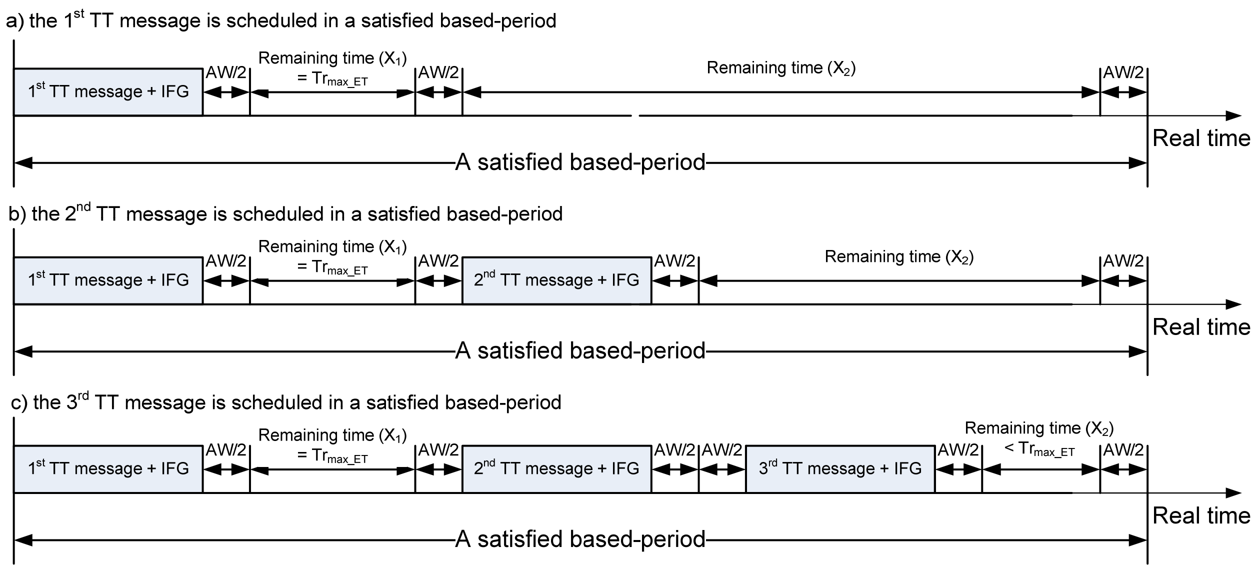


Fig. 5.14: Two unequal remaining times in a satisfied based-period in distributed offset form

With regard to TTE-network delays, the overall remaining time of a satisfied based-period is illustrated in Fig. 5.15 (X1+X2), and calculated using Equation 5.19. A remaining time between any two TT-messages can be obtained using Equation 5.18. If two consecutive TT-messages are transmitted from the same sender node, the TTE-network delay (ND_{TTE}) in Equation 5.19 will be zero. Otherwise, the TTE-network delay in Equation 5.19 is calculated using Equation 5.21.

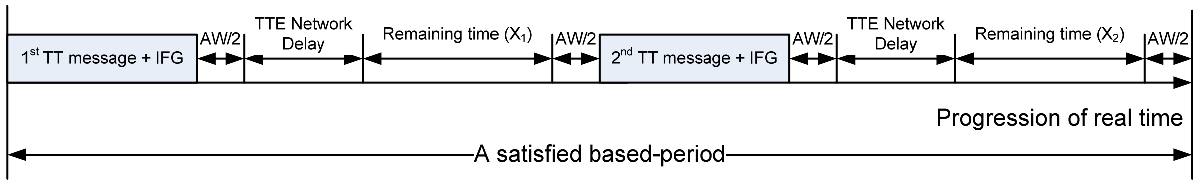


Fig. 5.15: Remaining times in distributed offset form in consideration of TTE network delays

$$X = SBP - \{Tr_{TT} + N * (AW + IFG) + ND_{TTE}\} \text{ , ...(5.19)}$$

$$Tr_{TT} = \frac{1}{BW}\sum_{i=0}^{n} msg\_size_i \text{ , ...(5.20)}$$

$$ND_{TTE} = \sum_{j=1}^{N}\left(DD_{sender(j)} + TT_{msg(j)} + D_{IFG(j)} + PD_{sender(j)} + \sum_{i=1}^{N_{sw}} LSW_{(i,j)}\right)\text{,.............(5.21)}$$

$$LSW_{(i,j)} = TT_{msg(i,j)} + D_{IFG(i,j)} + PD_{(i,j)} + TTL_{(i,j)} + DD_{(i,j)} \text{ ,.......................................(5.22)}$$

, where

X is the overall remaining time for standard-Ethernet traffic in a satisfied based-period,

SBP is the satisfied based-period of a TT-communication schedule,

Tr_{TT} is the summation of all TT-message transmission time in the satisfied based-period,

N is the number of TT messages in the satisfied based-period,

IFG is the time interval of an inter-frame gap,

AW is the acceptance window of TTEthernet,

BW is the Ethernet bandwidth,

ND_{TTE} is the TTE-network delay of all TT messages in the satisfied based-period,

msg_size_i is the size of the ith frame in bits within the satisfied based-period,

$DD_{sender(j)}$ is the dispatch delay of the jth TT message at a sender node,

$TT_{msg(j)}$ is the transmission time of the jth TT message at the sender node,

$D_{IFG(j)}$ is the time interval of an inter-frame gap of the jth TT message at the sender node,

$PD_{sender(j)}$ is the propagation delay of an Ethernet network cable connected to the sender node which transmits the jth TT message,

$LSW_{(i,j)}$ is the latency of the ith TTE switch for relaying the jth TT message,

$TT_{msg(i,j)}$ is the transmission time of the jth TT message at the ith TTE switch,

$D_{IFG(i,j)}$ is an inter-frame gap (IFG) of the jth TT message at the ith TTE switch,

$LSW_{(i,j)}$ is the latency of the jth TTE switch for relaying the ith TT message,

N_{SW} is the total number of TTE switches which the jth TT messages passes through from the sender node to a receiver node,

$PD_{(i,j)}$ is the propagation delay of an Ethernet network cable where the jth TT message uses to transmit out at the ith TTE switch,

$TTL_{(i,j)}$ is the TT latency of the ith TTE switch for the jth TT message,

$DD_{(i,j)}$ is the dispatch delay of the ith TTE switch for the jth TT message.

5.3.2.2 A time interval between two consecutive TT messages in a satisfied based-period

In this section we analyze a time interval between two consecutive messages (i.e. between two consecutive TT messages or between a TT message and a PCF) within a satisfied based-period in distributed offset form. In this analysis, we assume that a TT message is forwarded by a TTE switch in multicast form (i.e. a TT message received at a TTE switch is duplicated and then forwarded to all other ports of the TTE switch). With regard to TT-message transmission, a time interval between two consecutive TT-message's dispatch points in time depends on the sender node(s) of the two consecutive TT-messages. Therefore, there are two scenarios of TT-message transmission in distributed offset form with regard to sender nodes. These are described as follows.

1. *Two consecutive TT-messages sent from the same sender node:* The second TT-message of the two consecutive TT-messages can be transmitted after the transmission of the first TT-message (at the sender node) is complete, as shown in Fig. 5.16. The time interval (TI_{TT}) between the two consecutive TT-message's dispatch points in time can be calculated using Equation 5.23. In this case, the TTE-network delay of the first TT message is not taken into account.

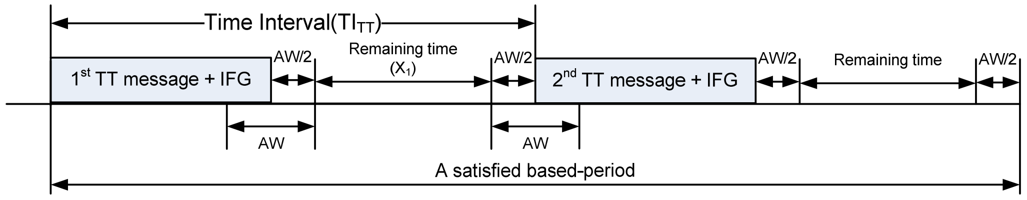


Fig. 5.16: Two consecutive TT-messages transmitted from the same sender

in distributed offset form

$$TI_{TT} = TT_{TT\_msg} + D_{IFG} + D_{AW} + X_1, \qquad (5.23)$$

, where TI_{TT} is the time interval between two consecutive TT-message's dispatch points in time, TT_{TT_msg} is the transmission time of the first TT message of two consecutive TT-messages, D_{IFG} is a time interval of an inter-frame gap, and D_{AW} is the acceptance window of TTEthernet , X_1 is the remaining time between two consecutive TT-messages.

2. *Two consecutive TT-messages sent from two different sender nodes*: We call a node transmitting the first and second TT-message of two consecutive TT-messages the *first node* and *second node*, respectively. We assume that TT messages are transmitted in multicast form. Thus, the second TT-message can be transmitted from the second node after the first message from the first node has completely arrived at the second node. The time interval (TI_{TT}) between the two consecutive TT-messages depends on TTE-network delays between the first and second nodes, as shown in Fig. 5.17. The time interval between the two consecutive TT-message's dispatch points in time from different sender nodes is derived from Equation 5.24.

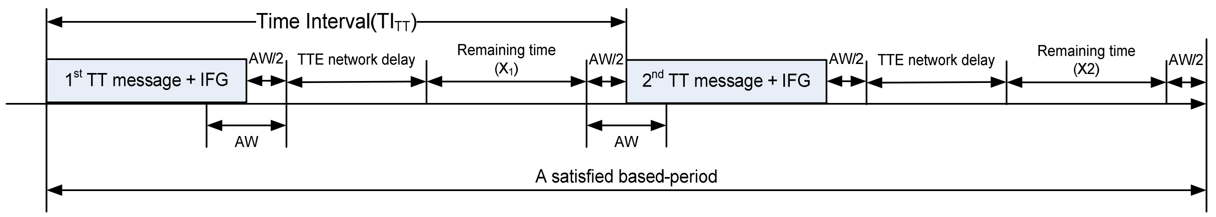


Fig. 5.17: Two consecutive TT-messages from two different senders in distributed offset form

$$TI_{TT} = TT_{TT\_msg} + D_{IFG} + D_{AW} + ND_{TTE} + X_1 \text{ , } \qquad (5.24)$$

$$ND_{TTE} = DD_{sender} + PD_{sender} + \sum_{i=1}^{N_{sw}} LSW_i \text{ ,} \qquad (5.25)$$

$$LSW_i = TT_{TT\_msg(i)} + D_{IFG(i)} + PD_i + TTL_i + DD_i \text{ ,} \qquad (5.26)$$

, where

TI_{TT} is the time interval between two consecutive TT-message's dispatch points in time,

TT_{TT_msg} is the transmission time of the first TT-message of two consecutive TT-messages,

D_{IFG} is a time interval of an inter-frame gap of the first TT-message at a sender node of the first TT-message,

D_{AW} is the acceptance window of TTEthernet,

ND_{TTE} is the TTE-network delay between the first node (the sender node of the first TT message) and the second node (the sender node of the second TT message),

X_1 is the remaining time between two consecutive TT-messages,

DD_{sender} is the dispatch delay of the first TT-message at the sender node of the first TT-message,

PD_{sender} is the propagation delay of an Ethernet network cable connected to the sender node of the first TT-message,

LSW_i is the latency of the ith TTE switch for relaying the first TT message,

N_{SW} is the total number of TTE switches which the first TT messages passes through between the sender node of the first TT-message and the sender node of the second TT-message,

$TT_{TT\_msg(i)}$ is the transmission time of the first TT message at the ith TTE switch,

$D_{IFG(i)}$ is the time interval of an inter-frame gap (IFG) at the ith TTE switch,

PD_i is the propagation delay of an Ethernet network cable where the first TT message is transmitted from the ith TTE switch,

TTL_i is the TT latency of the ith TTE switch for a TT message,

DD_i is the dispatch delay of TTEthernet at the ith TTE switch.

5.3.2.3 A time interval between a PCF and TT message in a satisfied based-period

According to the TTEthernet specification [TTE08], the offset of PCF traffic in a TT communication schedule is defined as zero ($Offset_{PCF} = 0$). TT messages can be transmitted only after all TTE devices within a TTE cluster have completely received their PCFs, as depicted in Fig. 5.18. A PCF travels from a synchronization master to a compression master in a TTE cluster. The compression master compresses the received PCF and then distributes it to all synchronization masters and synchronization clients in the same TTE cluster. The time it takes for a PCF to be transmitted from a synchronization master to the compression master and back to a node (TD_{PCF}) is calculated using Equation 5.3. The compression master delay of TTEthernet is the time it takes for a compression master to relay a PCF. It is described in detail in the TTEthernet specification [TTE08]. The time interval between a PCF and a TT message can be calculated using Equation 5.27.

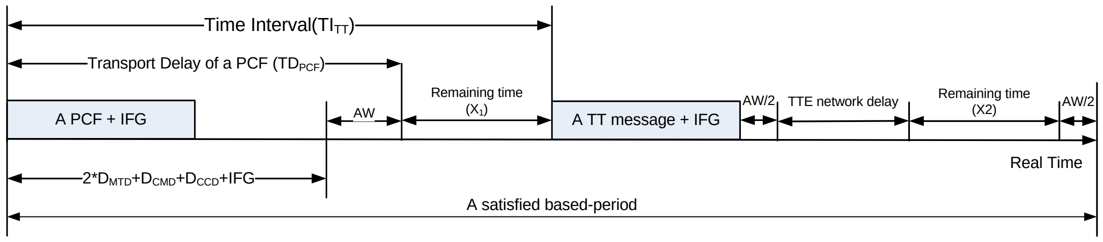


Fig. 5.18: The time interval between a PCF and a TT message in distributed offset form

$$TI_{PT} = TD_{PCF} + X_1 \text{ , } \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(5.27)}$$

, where TI_{PT} is the time interval between a PCF and TT message's dispatch points in time, TD_{PCF} is the PCF transport delay from a synchronization master to a compression master and back to any node in the same TTE cluster (as calculated using Equation 5.3), and X_1 is the remaining time between a PCF and TT message.

5.3.2.4 Offsets in distributed offset form

In a TT-communication schedule, the offset of PCF traffic is always zero. In Fig. 5.19, the first offset of TT traffic ($Offset_{TT\_1}$) in a TT-communication schedule is equal to the time interval between a PCF and a TT message (TI_{PT}), as derived from Equation 5.27. The second offset of TT traffic ($Offset_{TT\_2}$) is equal to the first offset of TT traffic ($Offset_{TT\_1}$) plus the time interval between the two TT message's dispatch points in time (TI_{TT}), as derived from Equation 5.23 or Equation 5.24. All offsets of TT traffic based on distributed offset form in a TTE cluster cycle can be obtained using Equation 5.28.

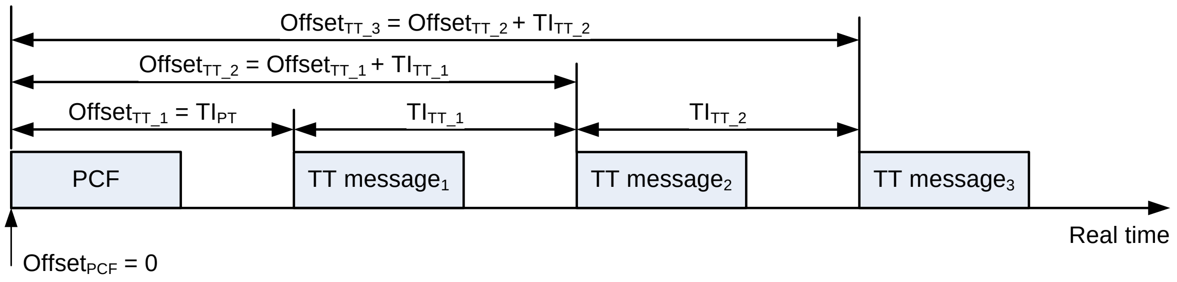

$Offset_{TT\_3}$ = $Offset_{TT\_2}$ + TI_{TT_2}
$Offset_{TT\_2}$ = $Offset_{TT\_1}$ + TI_{TT_1}
$Offset_{TT\_1}$ = TI_{PT}
TI_{TT_1}
TI_{TT_2}
PCF
TT message$_1$
TT message$_2$
TT message$_3$
Real time
$Offset_{PCF}$ = 0

Fig. 5.19: The offsets of a PCF and three TT messages in a TT-communication schedule

$$Offset_{TT\_(j)} = TI_{PT} + \sum_{i=1}^{j-1} TI_{TT\_(i-1)} \text{ ,………..………….…………….………….....(5.28)}$$

, where $Offset_{TT\_(j)}$ is the offset of TT traffic of the ith time-critical application.

TI_{PT} is the time interval between a PCF and TT message's dispatch points in time, as calculated using Equation 5.27,

$TI_{TT\_(i-1)}$ is the time interval between the (i-1)th TT-message and the ith TT-message points in time, as derived from Equation 5.23 or Equation 5.24. Note that $TI_{(0)}$ is equal to zero.

5.3.2.5 Distributed offset form in a TTE cluster cycle

In the previous sections we have introduced the fundamental concept of distributed offset form. In this section we extend this concept into a TTE-cluster cycle. Based on distributed offset form, we can obtain the offsets of time-critical applications in a TTE-cluster cycle, by using the method described below. In this method, one of given time-critical applications takes responsibility to generate PCFs (synchronization messages). Our method is illustrated with four time-critical applications. We call these applications *appl_1*, *appl_2*, *appl_3* and *appl_4*. They have periods of 2, 4, 4, and 6 milliseconds, respectively. The Ethernet payload-data size of a TT message for the four applications is 100, 46, 500, and 1000 bytes, respectively.

1) *Sort the given time-critical applications:* We sort the given time-critical applications using two parameters: periods and Ethernet payload-data sizes of time-critical applications. Firstly, the time-critical applications are sorted in ascending order with regard to their periods. If there is a tie, they will then be sorted using their Ethernet payload-data sizes in descending order. The first time-critical application of the sorted time-critical applications has the highest priority to be scheduled in a TTE cluster cycle. Note that the time-critical application with the lowest period will transmit a TT message in every time interval of a satisfied based-period along a TTE cluster cycle. In our example, the sorted order of the time-critical applications is: appl_1, appl_3, appl_2 and appl_4. The sorted time-critical applications will be scheduled in ascending order for the next step (i.e. appl_1 is the time-critical application to be considered first in the next step).

2) *Calculate the number of time-slots for a TTE-cluster cycle and time-critical applications:* The lowest period of time-critical applications in a schedule is the satisfied based-period of the schedule. A TTE-cluster cycle is equal to the Least Common Multiple (LCM) of all periods of time-critical applications in a schedule. The TTE-cluster cycle also denotes a number of satisfied based-periods. We call a time interval of a satisfied based-period a *time-slot*. A TTE-cluster cycle consists of a set of the LCM number of connected time-slots. We call all connected time-slots in a TTE-cluster cycle *cluster-cycle time-slots,* as depicted in Fig. 5.20. The number of time-slots in a TTE-cluster cycle is denoted as $NTS_{cluster}$. In our example, the lengths of a TTE-cluster cycle and the time-slot are equal to 12, and 2 milliseconds, respectively. Therefore, the number of the cluster-cycle time-slots ($NTS_{cluster}$) is 6. We also define the number of time-slots for one period of a time-critical application in a schedule as the satisfied harmonic-period of the time-critical application divided by the satisfied based-period. We denote the number of time-slots for one period of a time-critical application as NTS_{appl}. In our example, the number of time-slots for appl_1's period (NTS_{appl_1}) is one. The number of time-slots for appl_2's period (NTS_{appl_2}) is two. The number of time-slots for appl_3's period (NTS_{appl_3}) is also two. The number of time-slots for appl_4's period (NTS_{appl_4}) is three. This denotes that one TT message from appl_1 is transmitted every time-slot. One TT message from appl_2 and appl_3 are transmitted every two time-slots. One TT message from appl_4 is transmitted every three time-slots.

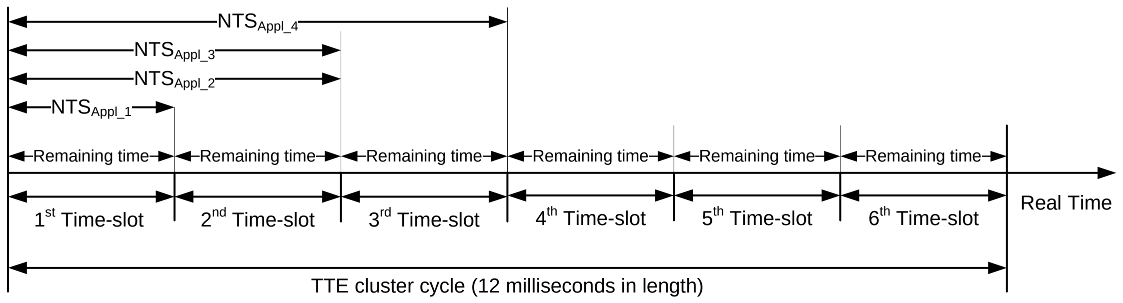


Fig. 5.20: The number of time-slots for time-critical applications

3) Put PCFs and TT messages from the sorted time-critical applications into time-slots: In this process we consider each of the sorted time-critical applications in order. That is, the 1st time-critical application in the sorted time-critical applications is considered first to put its messages into the time-slots. We put PCFs and TT messages from each of the sorted time-critical applications into time-slots using the following steps:

   Step 1: Find possible sets of time-slots for scheduling TT-messages: The number of possible sets of time-slots is equal to the number of time-slots for a time-critical application (NTS_{appl}). Due to the periodic transmission of TT traffic, a set of time-slots for scheduling TT-messages is defined using Equation 5.10. In our example, appl_1 has one time-slot ($NTS_{appl\_1} = 1$). Thus, appl_1 has the following set of time-slots: {1st, 2nd, 3rd, 4th, 5th, 6th}. For appl_2, NTS_{appl_2} is equal to two. Thus, there are two sets of time-slots: {1st, 3rd, 5th} and {2nd, 4th, 6th}. For appl_3, there are also two sets of time-slots: {1st, 3rd, 5th} and {2nd, 4th, 6th}. For appl_4 there are three sets of time-slots: {1st, 4th}, {2nd, 5th} and {3rd, 6th}.

   Step 2: Find the lowest remaining time of each set of time-slots for a time-critical application: The remaining time of each time-slot is calculated using Equation 5.8. The lowest remaining time of each set of time-slots is used in the next step.

   Step 3: Find a set of time-slots having the maximum of the lowest remaining-times of all sets of time-slots: We select the set of time-slots that has the maximum value of the lowest remaining time of all sets of time-slots. With such a set of time-slots for the time-critical application, TT messages and PCFs from that application are to be scheduled into the set of time-slots.

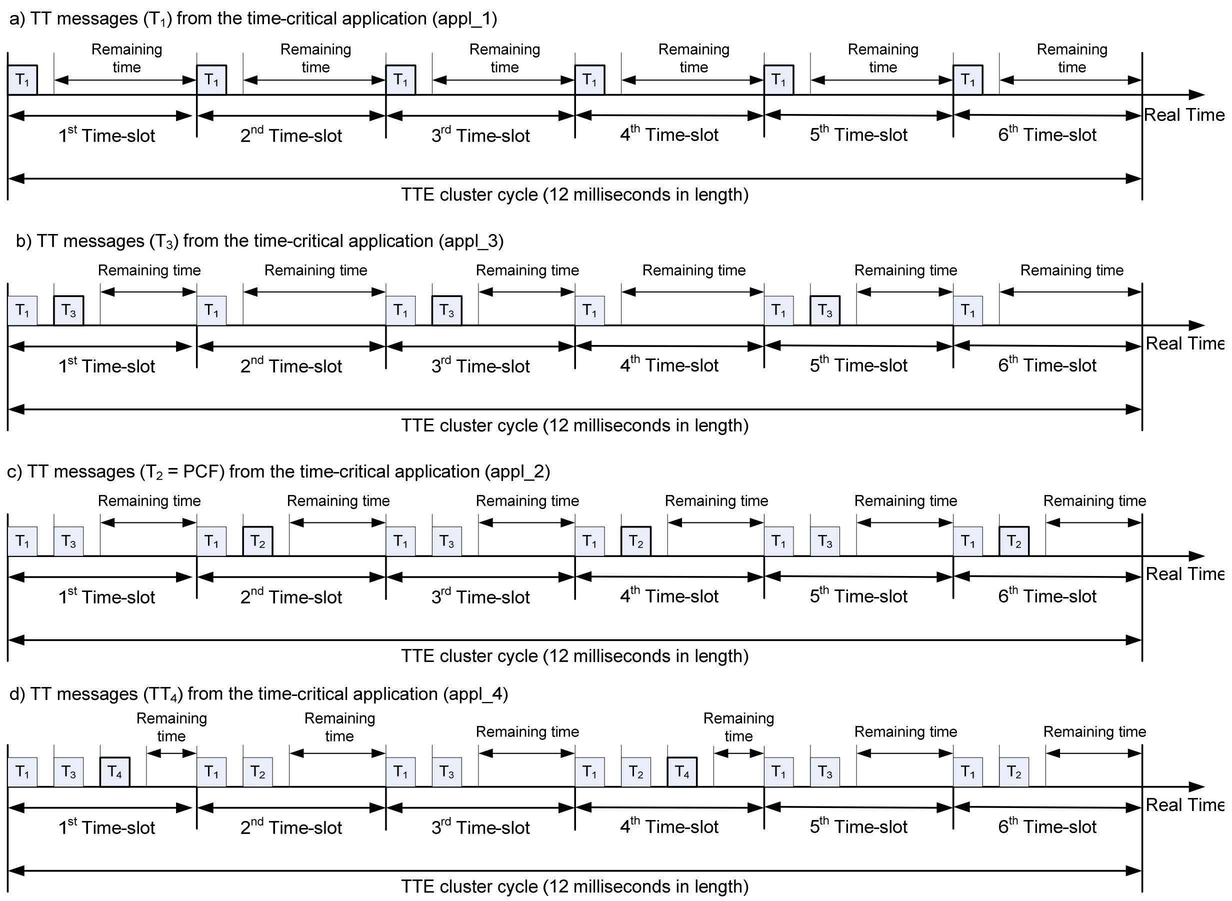

a) TT messages (T1) from the time-critical application (appl_1)
Remaining time
T1
1st Time-slot
2nd Time-slot
3rd Time-slot
4th Time-slot
5th Time-slot
6th Time-slot
Real Time
TTE cluster cycle (12 milliseconds in length)
b) TT messages (T3) from the time-critical application (appl_3)
T3
c) TT messages (T2 = PCF) from the time-critical application (appl_2)
T2
d) TT messages (TT4) from the time-critical application (appl_4)
T4

Fig. 5.21: TT messages contained in the time-slots of a TTE cluster cycle

In our example, NTS_{appl_1} is equal to one. There is one set of time-slots for appl_1, the 1st, 2nd, 3rd, 4th, 5th and 6th time-slots. A TT message is put in each of the sets of time-slots in continuous offset form, as shown in Fig. 5.21(a). For appl_3, NTS_{appl_3} is equal to two. There are two sets of time-slots for appl_3, the 1st, 3rd and 5th time-slots and the 2nd, 4th and 6th time-slots. After scheduling TT traffic from appl_1, the remaining time of each time-slot is the same, as depicted in Fig. 5.21(a). All sets of time-slots for appl_3 have an equal priority for being scheduled with TT traffic. We select the first set of time-slots to be scheduled with TT messages from appl_3 in continuous offset form, as depicted in Fig. 5.21(b). For appl_2, NTS_{appl_2} is equal to two. There are two sets of time-slots for appl_2, the 1st, 3rd and 5th time-slots and the 2nd, 4th and 6th time-slots. After scheduling TT traffic from appl_3, the lowest remaining time of the second set of time-slots is higher than that of the first set, as depicted in Fig. 5.21(b). Thus, the second set of time-slots is scheduled with PCFs from appl_2 in continuous offset form, as depicted in Fig. 5.21(c). Note that appl_2 is responsible for generating PCFs. For appl_4 there are three sets of time-slots, the 1st and 4th time-slots, 2nd and 5th time-slots and 3rd and 6th time-slots. The lowest remaining time for each set of time-slots is the same, as depicted in Fig. 5.21(c). We select the first set of time-slots for appl_4 to be scheduled with TT messages in continuous offset form, as depicted in Fig. 5.21(d).

4) Find the offsets of PCF and TT traffic in distributed offset form: In this process we distribute the remaining time of each time-slot in distributed offset form. The offsets of PCFs and TT messages can be obtained using the following steps:

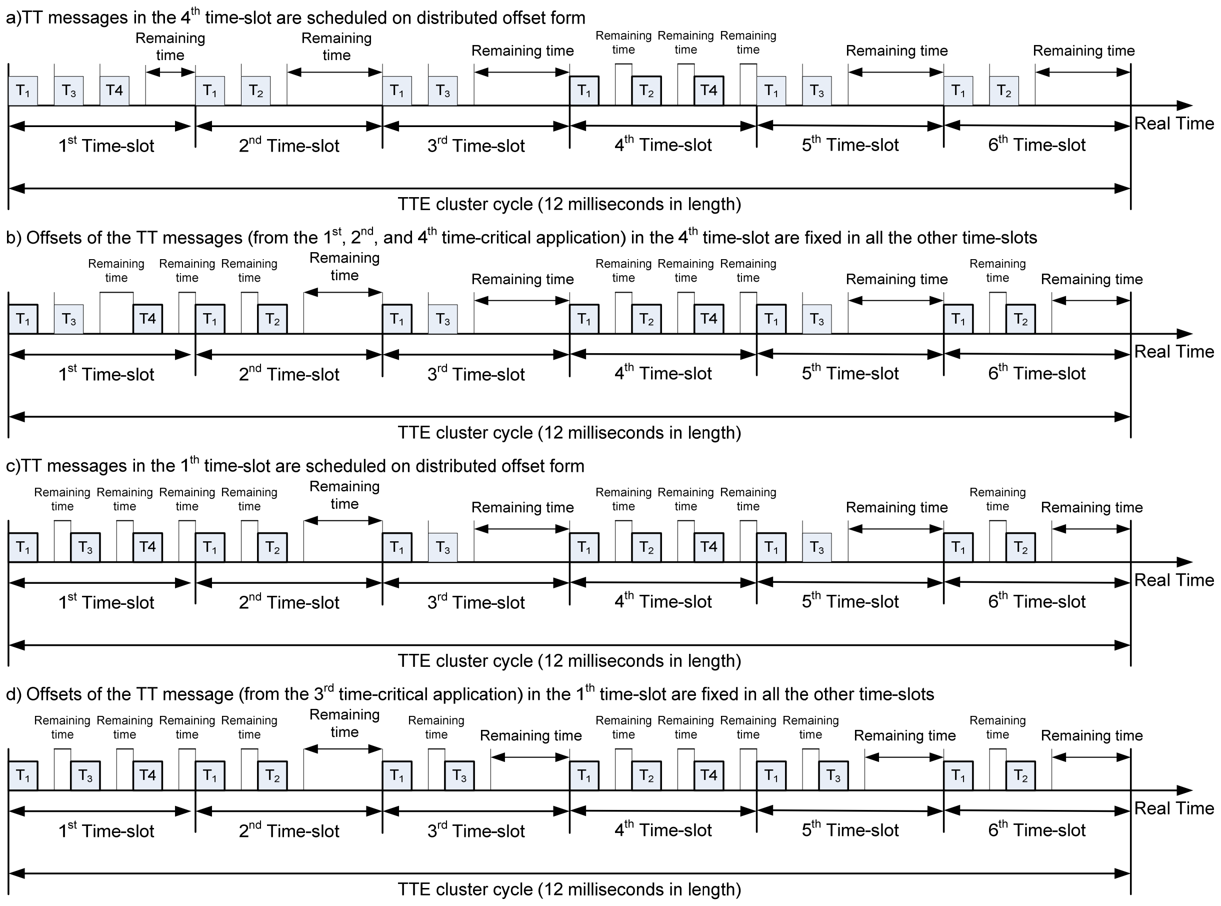

a)TT messages in the 4th time-slot are scheduled on distributed offset form
b) Offsets of the TT messages (from the 1st, 2nd, and 4th time-critical application) in the 4th time-slot are fixed in all the other time-slots
c)TT messages in the 1th time-slot are scheduled on distributed offset form
d) Offsets of the TT message (from the 3rd time-critical application) in the 1th time-slot are fixed in all the other time-slots
Remaining time
T1
T2
T3
T4
1st Time-slot
2nd Time-slot
3rd Time-slot
4th Time-slot
5th Time-slot
6th Time-slot
Real Time
TTE cluster cycle (12 milliseconds in length)

Fig. 5.22: The scheduling process based on distributed offset form in a TTE cluster cycle

Step 1: Schedule TT messages and PCFs in each time-slot in distributed offset form: In this step, all the time-slots from the previous process are considered in ascending order with regard to their remaining time. The time-slot with the lowest remaining time will be scheduled in distributed offset form first. The offsets of all messages which have already been scheduled in a time-slot will be the offsets of the messages (from the same time-critical applications) in other time-slots. In our example, we assume that all time-slots in ascending order of remaining time are the 4th, 1st, 2nd, 6th, 3rd and 5th time-slots. Fig. 5.22 depicts the scheduling result based on distributed offset form in this step.

Step 2: Calculate the offsets of PCF/TT traffic for all time-critical applications: The offsets of messages in a time-slot can be calculated using Equation 5.28. The concept behind this equation is depicted in Fig. 5.23. Note that appl_2 is responsible for generating PCFs.

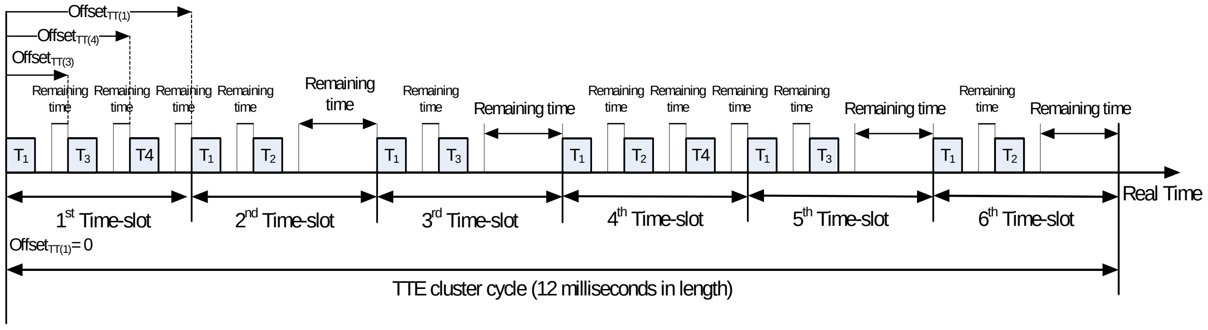


Fig. 5.23: The offsets of PCF and TT traffic in a TTE-cluster cycle (distributed offset form)

Step 3: Shift all the offsets with the time interval of the PCF offset: Due to the offset of PCF traffic in TTEthernet being defined as zero, all offsets from Step 2 are subtracted from the PCF offset (Offset$_{PCF}$), as shown in Equation 5.29. If the result of the subtraction (the new offset) is negative, the new offset will be added with the period of the time-critical application which the new offset belongs to. The result from shifting all offsets with the time interval of the PCF is depicted in Fig. 5.24.

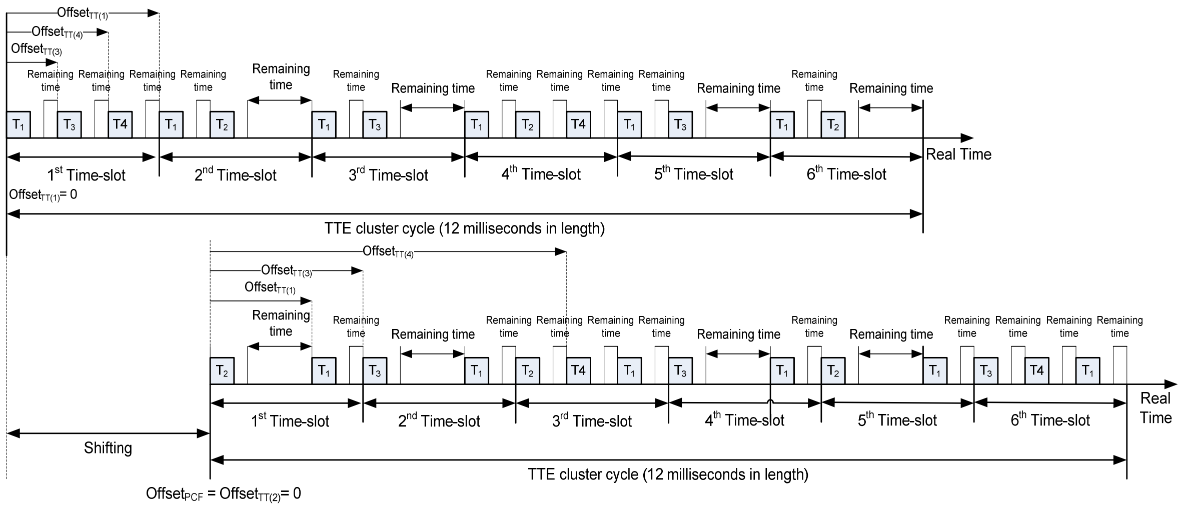


Fig. 5.24: The offsets of TT traffic after the shifting process (distributed offset form)

$$Offset_{TT\_(i)} = \begin{cases} Offset_{TT(i)} - Offset_{PCF}\,, \; if\big(Offset_{TT(i)} - Offset_{PCF} \geq 0\big) \\ Period_i + \;\; (Offset_{TT(i)} - Offset_{PCF}\,), \; otherwise \end{cases} \quad \ldots\ldots\ldots(5.29)$$

, where $Offset_{TT\_(i)}$ is the offset of TT traffic of the ith time-critical application after shifting,

$Offset_{TT(i)}$ is the offset of TT traffic of the ith time-critical application before shifting,

$Offset_{PCF}$ is the offset of PCF traffic before shifting,

$Period_i$ is the period of the i_{th} time-critical application,

5) Schedule PCF/TT traffic in a TTE cluster cycle: In the previous processes we obtain all offsets of time-critical applications. In this process, we calculate all dispatch instants of PCFs and TT messages along a TTE cluster cycle, using Equations 5.30 and 5.31, respectively. PCFs arrive at a TTE device at receiving points in time. These are calculated using Equation 5.32 (for a synchronization master or a synchronization client) and Equation 5.33 (for a compression master). TT messages are received by TTE devices at receiving points in time. These are calculated using Equation 5.34.

$$DIT_{PCF} = (n-1)(Period_{PCF}), n = \{1, 2, 3, \ldots, \frac{Cluster\ cycle}{Period_{PCF}}\} \ldots\ldots\ldots\ldots(5.30)$$

$$DIT_{TT(i)} = Offset_{TT(i)} + \;(n-1)\big(Period_{TT(i)}\big), n = \{1, 2, 3, \ldots, \frac{Cluster\ cycle}{Period_{PCF}}\} \ldots..(5.31)$$

$$RIT_{PCF(SM/SC)} = DIT_{PCF} + T_{PCF} + IFG + TD_{PCF}, \ldots\ldots(5.32)$$

$$RIT_{PCF(CM)} = DIT_{PCF} + T_{PCF} + IFG + MTD_{PCF}, \ldots\ldots(5.33)$$

$$RIT_{TT(i,j)} = DIT_{TT(i)} + T_{TT(i)} + IFG + ND_{TTE(i,j)}\ , \ldots\ldots(5.34)$$

, where DIT_{PCF} is the dispatch points in time for PCF traffic in a TTE cluster cycle,

$Period_{PCF}$ is the transmission period of PCF traffic,

$DIT_{TT(i)}$ is the dispatch points in time for TT traffic of the ith time-critical application in a TTE-cluster cycle,

$Offset_{TT(i)}$ is the offset of TT traffic of the ith time-critical application in a TTE cluster cycle.

T_{PCF} is the transmission time of a PCF = (PCF’s length)/(Ethernet bandwidth),

IFG is the time interval of an inter-frame gap,

TD_{PCF} is the PCF transport delay from a synchronization master to a compression master and back to any node in the same TTE cluster (Equation 5.3),

$RIT_{PCF(SM/SC)}$ is the receiving points in time for PCF messages at a synchronization master or a synchronization client,

MTD_{PCF} is the maximum transmission delay of a TTE system,

$RIT_{PCF(CM)}$ is the receiving points in time for PCF messages at a compression master,

$RIT_{TT(i,j)}$ is the receiving points in time for TT messages of the i^{th} time-critical application at the j^{th} TTE device,

$T_{TT(i)}$ is the transmission time of a TT message of the i^{th} time-critical application (TT message's length)/(Ethernet bandwidth),

$ND_{TTE(i,j)}$ is the TTE-network delay of a TT message of the i^{th} time-critical application from a source node to the j^{th} TTE device.

5.4 Summary

By using our TT-traffic scheduling approach, we can obtain various TT-communication schedules from (distributed) time-critical applications. In our TT-traffic scheduling approach, we propose the *remaining-bandwidth maximization approach* for obtaining optimal periods of time-critical applications. In this approach, we firstly enumerate possible schedules (periods only) from the periods of specified time-critical applications. Each possible schedule is then analyzed by calculating the amount of remaining bandwidth for standard-Ethernet traffic. The schedule with the highest remaining bandwidth for standard-Ethernet traffic is supposed to be the optimal one in term of throughput. We use the periods from the obtained schedules to find the offsets of the periods. In our TT-traffic scheduling approach, we classify offset forms into two possible forms: *continuous offset form* and *distributed offset form*. In both offset forms we have analyzed the maximum waiting time of standard-Ethernet traffic caused by TT-traffic transmission. We have found that a TT communication schedule based on distributed offset form results in less waiting time for standard-Ethernet traffic than one based on continuous offset form. Additionally, we have presented the principles of both offset forms and methodologies to apply them into a TTE cluster cycle. With both periods and offsets of specified time-critical applications, we can obtain the dispatch and receiving points in time for all TT messages and PCFs along the TTE cluster cycle. The TTE cluster cycle with such dispatch and receiving points in time is the TT-communication schedule for TTEthernet systems.

CHAPTER 6

Performance of standard-Ethernet Traffic in TTEthernet Systems

In this chapter we evaluate the performance of standard-Ethernet traffic in both simulated and actual 100-Mbps TTEthernet systems. We do this using a variety of confliction-free time-triggered (TT) communication schedules. This chapter begins by detailing the methods used to set up and configure a network structure for both a simulated and an actual TTEthernet network. By using our scheduling approach, all possible TT communication schedules are enumerated from the specified periods of given time-critical applications. We assign standard-Ethernet traffic to both types of TTEthernet system (simulated and real), in order to measure the performance achieved (throughput and latency). This chapter finishes with an analysis and summary of our obtained results.

6.1 Network structure and configuration

We define a TTEthernet network structure comprising a single TTE switch and the following three nodes: a standard-Ethernet frame sender node (the sender node for ET traffic), a node to generate TT messages (the sender node for TT traffic), and a receiver node. This network structure is depicted in Fig. 1.

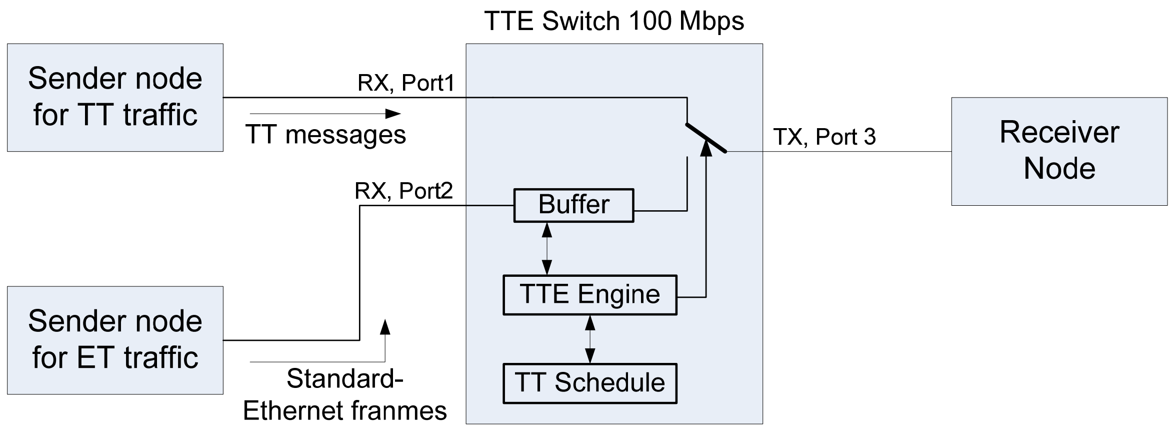


Fig. 6.1: Our TTEthernet network structure

6.1.1 The TTE switch: A 100-Mbps 8 port TTE switch containing a TT-communication schedule. With this schedule, the TTE switch can receive and dispatch TT messages at predefined points in time. In a TTE switch, the TT traffic takes precedence over the standard-Ethernet traffic. The TTE switch we use in the real TTEthernet network is available from TTTech [TTTech]. In the simulated TTE switch, we define the TTE switch's configuration as follows:

- The queue buffer for standard-Ethernet frames is 256,000 bytes.
- The transmission rate of standard-Ethernet traffic is 100 Mbps per port.
- The latency for relaying a TT message (TT latency) is 0.01 microseconds. This value is always constant.
- The latency for relaying a standard-Ethernet frame (ET latency) is 6.72 microseconds. This value is derived from the forwarding rate of an AT-FS750/16 16 port Fast Ethernet WebSmart Switch (148,800 pps) [All09].

6.1.2 The sender node for ET traffic: Used for transmitting Ethernet frames to the receiver node via the TTE switch. The sender node contains a 100-Mbps Ethernet controller interface. In the simulated TTEthernet network, the sender node cannot transmit TTEthernet traffic. Therefore the node transmits only standard-Ethernet traffic. The sender node for ET traffic used in the real TTEthernet network is a personal computer (Intel Pentium 4 CPU 2.80 GHz) with a Linux based operating system (Ubuntu version 2.6.24-19).

6.1.3 The sender node for TT traffic: A 100-Mbps TTEthernet end system used for transmitting TT messages to the receiver node via the TTE switch. TT messages are transmitted according to its TT communication schedule. We define this sender node as a synchronization master. In the simulated TTEthernet network, the sender node for TT traffic can only transmit TTEthernet traffic. The sender node for TT traffic used in the real TTEthernet network is available from TTTech [TTTech].

6.1.4 The receiver node: Used for receiving all messages (both TT messages and standard-Ethernet frames) from the TTE switch. In the real TTEthernet network, we use a laptop (Intel Core 2 T7200 2 GHz with a Linux based operating system – Ubuntu version 2.6.24-19) as the receiver node. It is also used to monitor traffic from the TTE switch, and measure standard-Ethernet performance.

6.1.5 The network cables: We use standard-Ethernet cables to connect the TTE switch and each node in a star network topology. In the simulated TTEthernet system, we define the length of each network cable as 5 meters. Therefore, the propagation delay of each network cable is 0.075 microseconds. Note that the speed of light in a copper cable is 2/3 times the speed of light in vacuum ($3 \text{x} 10^8$ m/s) [Kop97, pp. 160].

6.1.6 TTEthernet configuration: We set the configuration parameters of TTEthernet as follows:

- Bounded local precision: Set to 0.5 microseconds for the simulated TTEthernet system and 2 microseconds for the real TTEthernet system. The *acceptance window* is automatically equal to 2 times the *bounded local precision*. This makes the acceptance windows of the simulated TTEthernet system and the actual TTEthernet system 1 and 4 microseconds, respectively. The acceptance window parameter also denotes a bounded cluster precision of global time.
- Faulty master nodes: In our work, faulty master nodes are not considered.

6.2 Generating standard-Ethernet traffic

In Fig. 6.1, the sender node for ET traffic generates standard-Ethernet frames and sends them to the receiver node, via the TTE switch. All Ethernet frames are sent in unicast form. In our work, the sender node for ET traffic transmits standard-Ethernet frames into the TTEthernet network. The purpose of this is to measure the performance of the standard-Ethernet traffic. We define the pattern of generating standard-Ethernet traffic for both types of TTEthernet systems as follows:

6.2.1 **Generating standard-Ethernet traffic in the simulated TTEthernet system:** We define the patterns of generating standard-Ethernet traffic in the simulated TTEthernet system with respect to the following key performance parameters:

6.2.1.1 **Standard-Ethernet Throughput**: We define four experimental scenarios with respect to the following standard-Ethernet payload data sizes: 46, 512, 1024 and 1500 bytes. Each experimental scenario transmits Ethernet messages with the full bandwidth of the Ethernet controller for the ET traffic sender node . At the receiver node, the average throughput of the standard-Ethernet traffic is recorded.

6.2.1.2 **Transport delays of Ethernet frames**: Each of the four experimental scenarios described above have seven sub-scenarios, each with a different transmission rate of Ethernet payload data (10%, 20%, 30%, 40%, 50%, 60% and 70% of the 100-Mbps bandwidth). At the receiver node, the transport delays of the standard-Ethernet frames for each sub-scenario are recorded. In these scenarios, the inter-departure time between any two generated standard-Ethernet frames are random with a poisson distribution.

6.2.2 **Generating standard-Ethernet traffic in the physical TTEthernet system:** We make use of available software tools for generating standard-Ethernet traffic, and for measuring standard-Ethernet performance. The following are the key performance parameters for the physical TTEthernet system:

6.2.2.1 **UDP Throughput:** To measure the standard-Ethernet performance in a physical TTEthernet system, we utilize the open-source tool: Distributed Internet Traffic Generator (D-ITG) version 2.6.1d [DIGT] [BDP07]. More specifically, this tool is used for generating standard-Ethernet traffic, and measuring its average throughput. With D-ITG, we can define the standard-Ethernet traffic in the form of UDP traffic. The Ethernet payload data size of a UDP message is equal to the summation of the IP header length (20 bytes), the UDP header length (8 bytes), and the UDP message’s payload data size. To obtain the maximum throughput, we define the amount of generated UDP traffic as 100% of Fast Ethernet’s bandwidth. The UDP payload data size of each UDP message is 1,472 bytes, denoting 1,500 bytes of Ethernet payload data. The physical TTEthernet network structure is shown in Fig. 6.1.

6.2.2.2 **Round trip delays of standard-Ethernet frames:** To measure the timing performance of standard-Ethernet traffic in the physical TTEthernet system, we firstly need to generate such traffic. To do this we use the ICMP (Internet Control Message Protocol) request/reply, or the standard 'ping' command. We then measure the round trip delays. An ICMP message is encapsulated by the Internet Protocol (IP) and standard Ethernet. The Ethernet payload data size of an ICMP message is equal to the summation of the IP header length (20 bytes), ICMP header length (8 bytes), and ICMP payload data size. We define the ICMP payload data size of each ICMP message as 1,472 bytes, denoting 1,500 bytes of Ethernet payload data. The inter-departure time between any two ICMP messages is 200 microseconds (i.e. at around a standard-Ethernet traffic load of 61.52 Mbps, including the overhead and inter-frame gap).

6.3 Generating TT traffic

The sender node for TT traffic is responsible for generating TT traffic according to a predefined TT communication schedule. By using our scheduling approach, we obtain the various TT communication schedules from the specified periods of given time-critical applications. In our experiments, we run eight time-critical applications in the sender node for TT traffic. The given periods of these time-critical applications (appl_1, appl_2, appl_3, appl_4, appl_5, appl_6, appl_7 and appl_8) are 3, 3, 3, 4, 4, 4, 5 and 5 milliseconds, respectively. The Ethernet payload data sizes of these time-critical applications are 1,500, 1,400, 1,300, 1,500, 1,400, 1,300, 1,500 and 1,400 bytes, respectively. We define the resynchronization period of PCF traffic as 5 milliseconds. The Ethernet payload data size of a PCF is 46 bytes. Based on our scheduling approach, the scheduling result of both TT and PCF traffic in the simulated TTEthernet systems is shown in Table 6.1.

Table 6.1: The TT communication schedules for the simulated TTEthernet systems

Application	size	period	Offset form	1st schedule		2nd schedule		3rd schedule	
				Period	Offset (us)	Period	Offset (us)	period	Offset (us)
Appl_1	1500 bytes	3 ms	Con.	3 ms	2,063.68	2 ms	1,063.68	2.5 ms	1,563.68
			Dis.		236.16		125.04		217.72
Appl_2	1400 bytes	3 ms	Con.	3 ms	2,187.72	2 ms	1,187.72	2.5 ms	1,687.72
			Dis.		588.64		591.48		551.76
Appl_3	1300 bytes	3 ms	Con.	3 ms	2,303.76	2 ms	1,303.76	2.5 ms	1,803.76
			Dis.		933.12		924.92		877.8
Appl_4	1500 bytes	4 ms	Con.	3 ms	2,411.8	4 ms	3,411.80	2.5 ms	1,911.8
			Dis.		1,269.6		1,250.36		1,195.84
Appl_5	1400 bytes	4 ms	Con.	3 ms	2,535.84	4 ms	3,535.84	2.5 ms	2,035.84
			Dis.		1,622.08		1,591.8		1,529.88
Appl_6	1300 bytes	4 ms	Con.	3 ms	2,651.88	4 ms	3,651.88	2.5 ms	2,151.88
			Dis.		1,966.56		1,925.24		1,855.92
Appl_7	1500 bytes	5 ms	Con.	3 ms	2,759.92	4 ms	3,759.92	5 ms	4,759.92
			Dis.		2,303.04		3,275.28		2,278.82
Appl_8	1400 bytes	5 ms	Con.	3 ms	2,883.96	4 ms	3,883.96	5 ms	4,883.96
			Dis.		2,655.52		3,641.64		4,673.96
PCF	46 bytes	5 ms	Con.	3 ms	0 us	4 ms	0 us	5 ms	0 us
			Dis.		0 us		0 us		0 us
Remaining Bandwidth for standard-Ethernet traffic				69,024 kbps		68,188 kbps		67,686.4 kbps	

Note that Con. and Dis. denote continuous offset form and distributed offset form, respectively.

In a physical 100-Mbps TTEthernet system, the minimum time interval between the dispatch instants of any two consecutive TT messages is bounded to be not less than 200 microseconds. Thus, standard-Ethernet frames can be transmitted between any two consecutive TT messages. In other words, we cannot schedule the given time-critical applications in continuous offset form in our physical TTEthernet system. However, by using a constraint where there is a minimum time interval between the dispatch instants of any two consecutive TT messages, we can utilize continuous offset form. The scheduling result for our physical TTEthernet system is shown in Table 6.2.

Table 6.2: The TT communication schedules for the actual TTEthernet systems

Application	size	period	Offset form	1st schedule		2nd schedule		3rd schedule	
				Period	offset	period	offset	period	Offset
Appl_1	1500 bytes	3 ms	Con.	3 ms	1400 us	2 ms	400 us	2.5 ms	900 us
			Dis.		200 us		250 us		750 us
Appl_2	1400 bytes	3 ms	Con.	3 ms	1600 us	2 ms	600 us	2.5 ms	1100 us
			Dis.		550 us		600 us		1000 us
Appl_3	1300 bytes	3 ms	Con.	3 ms	1800 us	2 ms	800 us	2.5 ms	1300 us
			Dis.		900 us		950 us		1250 us
Appl_4	1500 bytes	4 ms	Con.	3 ms	2000 us	4 ms	3000 us	2.5 ms	1500 us
			Dis.		1250 us		3300 us		1500 us
Appl_5	1400 bytes	4 ms	Con.	3 ms	2200 us	4 ms	3200 us	2.5 ms	1700 us
			Dis.		1600 us		1300 us		1750 us
Appl_6	1300 bytes	4 ms	Con.	3 ms	2400 us	4 ms	3400 us	2.5 ms	1900 us
			Dis.		1950 us		1650 us		2000 us
Appl_7	1500 bytes	5 ms	Con.	3 ms	2600 us	4 ms	3600 us	5 ms	4600 us
			Dis.		2300 us		3650 us		2250 us
Appl_8	1400 bytes	5 ms	Con.	3 ms	2800 us	4 ms	3800 us	5 ms	4800 us
			Dis.		2650 us		2000 us		4750 us
PCF	46 bytes	5 ms	Con.	3 ms	0 us	4 ms	0 us	5 ms	0 us
			Dis.		0 us		0 us		0 us
Remaining Bandwidth for standard-Ethernet traffic				69,024 kbps		68,188 kbps		67,686.4 kbps	

Note that Con. and Dis. denote continuous offset form and distributed offset form, respectively.

6.4 Performance analysis of standard-Ethernet traffic in TTE systems

In this section we describe the results of running both the simulated and physical TTEthernet systems. We present the performance metrics regarding standard-Ethernet traffic for each type of system.

6.4.1 The throughputs of standard-Ethernet traffic in the simulated TTE systems

To measure standard-Ethernet throughput in the simulated TTEthernet systems, we use four experimental scenarios of standard-Ethernet traffic loads, with respect to the following Ethernet payload data sizes: 46, 512, 1024 and 1500 bytes. Each experimental scenario has seven sub-scenarios, containing the following TT communication schedules: (1) no TT schedule, (2) the first schedule in continuous offset form, (3) the first schedule in distributed offset form, (4) the second schedule in continuous offset form, (5) the second schedule in distributed offset form, (6) the third schedule in continuous offset form, and (7) the third schedule in distributed offset form. In each experimental scenario, the sender node for ET traffic transmits standard-Ethernet messages with the full available bandwidth of its Ethernet controller. The throughputs of standard-Ethernet traffic in the simulated TTEthernet system is given in Table 6.3, and depicted in Fig. 6.2.

Table 6.3: The throughputs of standard-Ethernet traffic in the simulated TTEthernet systems

Contents	Throughputs (bits per second) of Ethernet frames			
Ethernet payload data sizes	46 bytes	512 bytes	1024 bytes	1500 bytes
(1) No TT schedule	54,761,904.76	93,090,909.09	96,421,845.57	97,529,258.78
(2) First schedule in COF	37,413,333.33	62,805,333.33	65,536,000.00	64,000,000.00
(3) First schedule in DOF	36,432,000.00	61,440,000.00	49,152,000.00	36,000,000.00
(4) Second schedule in COF	36,984,000.00	61,440,000.00	63,488,000.00	63,000,000.00
(5) Second schedule in DOF	36,616,000.00	54,272,000.00	53,248,000.00	36,000,000.00
(6) Third schedule in COF	36,726,400.00	61,440,000.00	63,897,600.00	62,400,000.00
(7) Third schedule in DOF	36,432,000.00	54,067,200.00	52,428,800.00	40,800,000.00

Note that COF and DOF are the abbreviations of Continuous Offset Form and Distributed Offset Form, respectively.

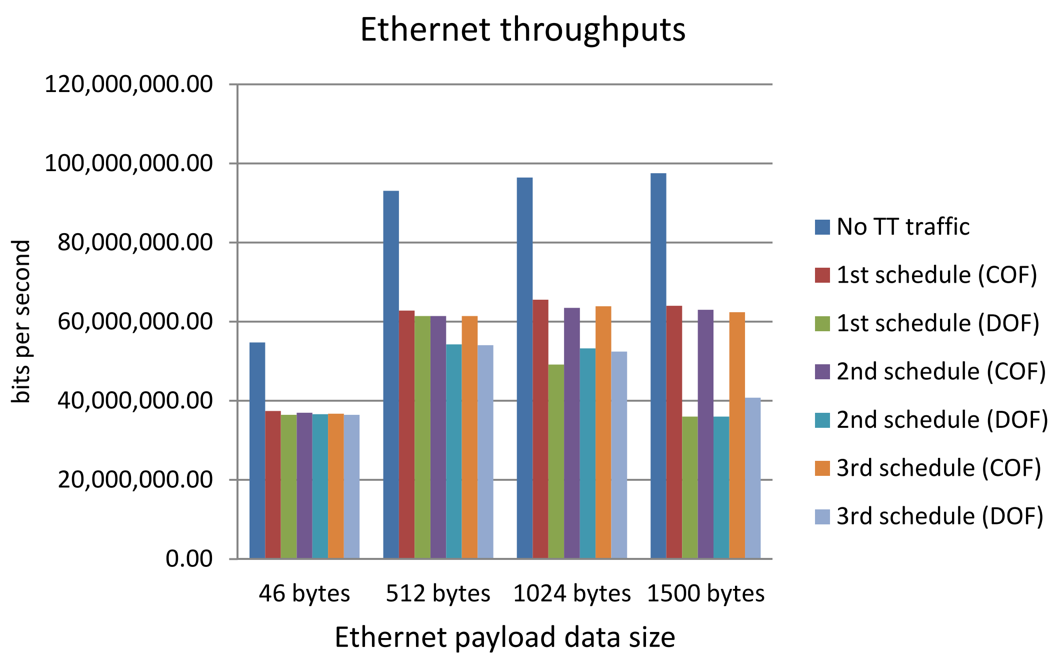


Fig. 6.2: The throughputs of standard-Ethernet traffic in the simulated TTEthernet systems

Throughput analysis of standard-Ethernet traffic: Using the simulation results, we now analyze the standard-Ethernet traffic throughputs.

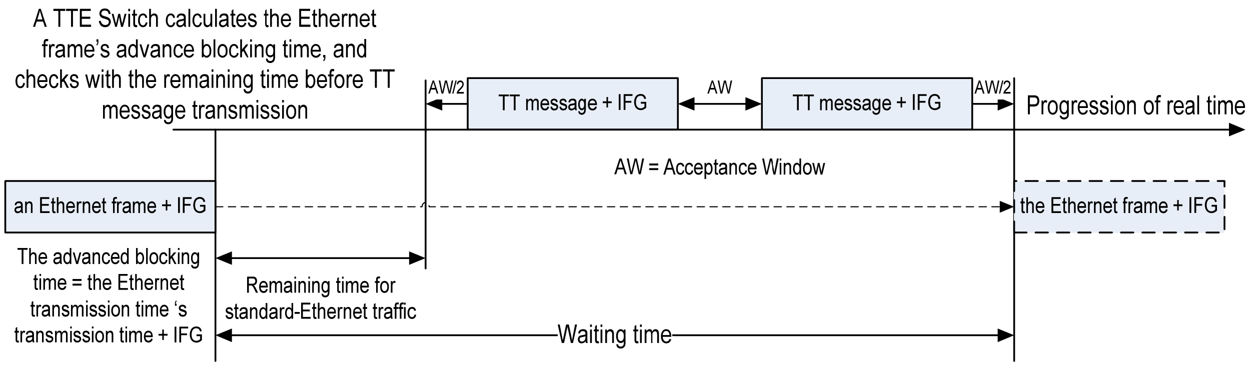

A TTE Switch calculates the Ethernet frame's advance blocking time, and checks with the remaining time before TT message transmission
AW/2
TT message + IFG
AW
TT message + IFG
AW/2
Progression of real time
AW = Acceptance Window
an Ethernet frame + IFG
the Ethernet frame + IFG
The advanced blocking time = the Ethernet transmission time 's transmission time + IFG
Remaining time for standard-Ethernet traffic
Waiting time

Fig. 6.3: The transmission of a standard-Ethernet frame in the simulated TTE switch

(1) Schedules in continuous offset form give higher standard-Ethernet throughputs than the corresponding schedules in distributed offset form. For example, in Table 6.3, with regard to the 1,500 byte data size, the standard-Ethernet throughput in the first schedule in continuous offset form is higher than that for the first schedule in distributed offset form (64 versus 36 Mbps). This is because a standard-Ethernet frame in the simulated TTE switch cannot be transmitted while a TT message is being transmitted, as depicted in Fig. 6.3. In other words, once a standard-Ethernet frame arrives at the transmitter of a TTE-switch port, the TTEthernet engine checks whether its transmission will fall in the forthcoming TT-message transmission interval. If the TTEthernet engine finds that the transmission of the standard-Ethernet frame will fall in the forthcoming TT-message transmission interval, the standard-Ethernet frame has to wait in the queue buffer until the TT-message transmission has finished. This implies that a standard-Ethernet frame can pass through a switch if it arrives at the transmitter of the TTE-switch port before any dispatching point in time for TT messages. This is calculated using *the time interval of the standard-Ethernet frame's transmission time plus the inter-frame gap of standard-Ethernet,* as depicted in Fig 6.3. We call this time interval the *advanced blocking time for TT traffic*. The advanced blocking time for TT traffic depends on the size of the arriving standard-Ethernet frame, as depicted in Fig 6.3. If the size of this frame is large, the advanced blocking time for TT traffic increases. A schedule in distributed offset form introduces a higher amount of advanced blocking time for TT traffic within its TTE cluster cycle than the corresponding schedule in continuous offset form. Therefore, a schedule in continuous offset form provides higher standard-Ethernet throughput than its corresponding schedule in distributed offset form.

(2) The first schedule in continuous offset form gives the highest standard-Ethernet throughput in all schedules (other than where there is no TT schedule), as shown in Table 6.3. Schedules in continuous offset form provide sets of closely consecutive TT-message transmissions, whereas schedules in distributed offset form evenly distribute TT-message transmissions. Therefore, the advanced blocking time for TT traffic in a switch using a schedule in continuous offset form is less than one using the corresponding schedule in distributed offset form. Thus, standard-Ethernet traffic using a schedule in continuous offset form can highly utilize the bandwidth remaining after TT traffic transmission. The standard-Ethernet throughputs from all schedules in continuous offset form in the maximum Ethernet payload data size (1,500 bytes) scenario correspond to the results generated by the *remaining-bandwidth maximization* approach, as shown in Table 6.2. As shown in the table, for the simulated TTEthernet system with the maximum Ethernet payload data size (1500 bytes), the first and third schedules in continuous offset form result in the highest (64,000,000 bps) and lowest (62,400,000) standard-Ethernet throughputs, respectively. In the results for the remaining bandwidth for standard-Ethernet traffic according to Table 6.2, the first and third schedules give the highest and lowest remaining-bandwidth for standard-Ethernet traffic (69,024 kbps and 67,686.4 kbps, respectively).

(3) Throughputs of standard-Ethernet traffic with schedules in distributed offset form depend on Ethernet payload data size. It can be seen in the case of 512 bytes of payload data size, that the first schedule in distributed offset form results in the highest standard-Ethernet throughput with respect to all the schedules in distributed offset form. For the cases of 46 and 1024 bytes of payload data size, the second schedule in distributed offset form gives the highest standard-Ethernet throughput. In the case of the maximum Ethernet payload data size (1500 bytes), the third schedule in distributed offset form results in the highest standard-Ethernet throughput. Therefore, standard-Ethernet throughput in the simulated TTEthernet network using a schedule in distributed offset form not only depends on the amount of TT traffic, but also on the Ethernet payload data size.

(4) The first schedule in continuous offset form is the optimal TT communication schedule with regard to standard-Ethernet throughput. This is because it provides the highest throughput of standard-Ethernet traffic in the simulated TTEthernet network in all scenarios of Ethernet payload data size. We have explained why a schedule in continuous offset form gives a small advanced blocking time for TT traffic in comparison to the corresponding schedule in distributed offset form.

6.4.2 Transport delays of standard-Ethernet traffic in the simulated TTE systems

For the transport delay measurement of standard-Ethernet traffic in the simulated TTEthernet systems, we have setup four experimental scenarios of standard-Ethernet traffic loads with respect to various Ethernet payload data sizes (i.e. 46, 512, 1024 and 1500 bytes). Each experimental scenario has sub-scenarios that vary the standard-Ethernet traffic load, as detailed in Section 6.2.1.2. We measure and record the transport delays of each incoming standard-Ethernet frame at the receiver node. The transport delay of a standard-Ethernet frame is a time delay starting from the point in time when the standard-Ethernet frame is transmitted from the sender node, to the point in time when the entire standard-Ethernet frame is received at the receiver node. The simulation results are concluded with the average, standard deviation, and maximum of all transported standard-Ethernet frame's transport delays, as shown in the following tables and figures.

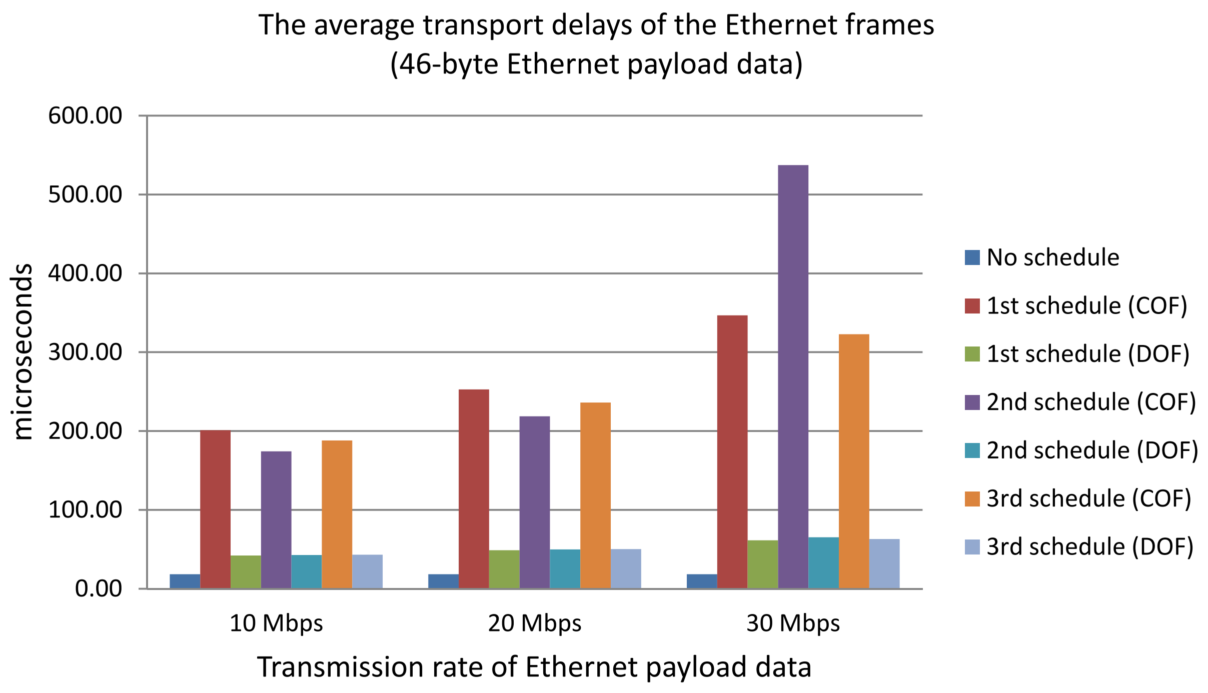


Fig. 6.4: The average transport delays of the standard-Ethernet traffic (46-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.4: The average transport delays of the standard-Ethernet traffic (46-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The average transport delays of the standard-Ethernet Ethernet traffic with 46-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	18.39	200.96	42.08	174.20	42.81	188.12	43.22
20 Mbps	18.39	252.87	48.82	218.73	49.81	236.17	50.35
30 Mbps	18.39	346.82	61.35	537.46	65.35	322.81	63.14
40 Mbps	18.39	BOF	BOF	BOF	BOF	BOF	BOF
50 Mbps	18.39	BOF	BOF	BOF	BOF	BOF	BOF
60 Mbps	18.39	BOF	BOF	BOF	BOF	BOF	BOF
70 Mbps	BOF	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

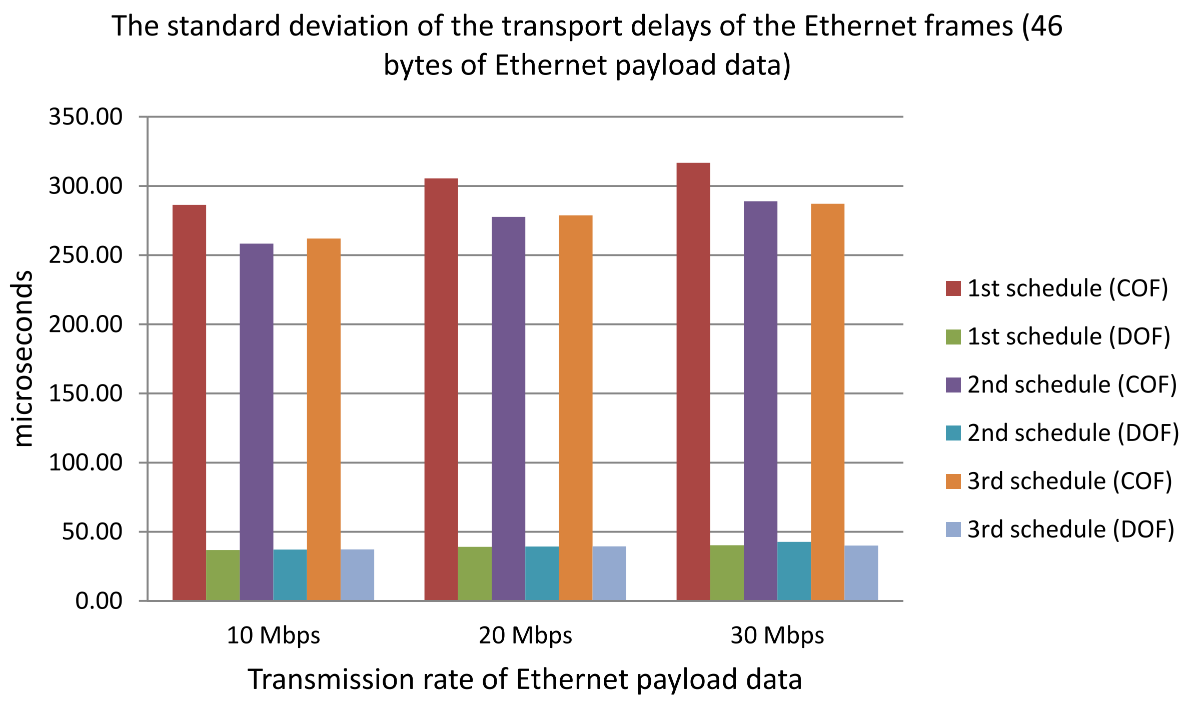


Fig. 6.5: The standard deviations of the standard-Ethernet traffic (46-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.5: The standard deviations of the transport delays of the standard-Ethernet traffic (46-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The standard deviation of the transport delays of the standard-Ethernet traffic with 46-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	0.00	286.34	36.81	258.31	37.14	261.96	37.30
20 Mbps	0.00	305.47	39.06	277.66	39.35	278.77	39.45
30 Mbps	0.00	316.70	40.29	288.98	42.76	287.13	39.99
40 Mbps	0.00	BOF	BOF	BOF	BOF	BOF	BOF
50 Mbps	0.00	BOF	BOF	BOF	BOF	BOF	BOF
60 Mbps	0.00	BOF	BOF	BOF	BOF	BOF	BOF
70 Mbps	BOF	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

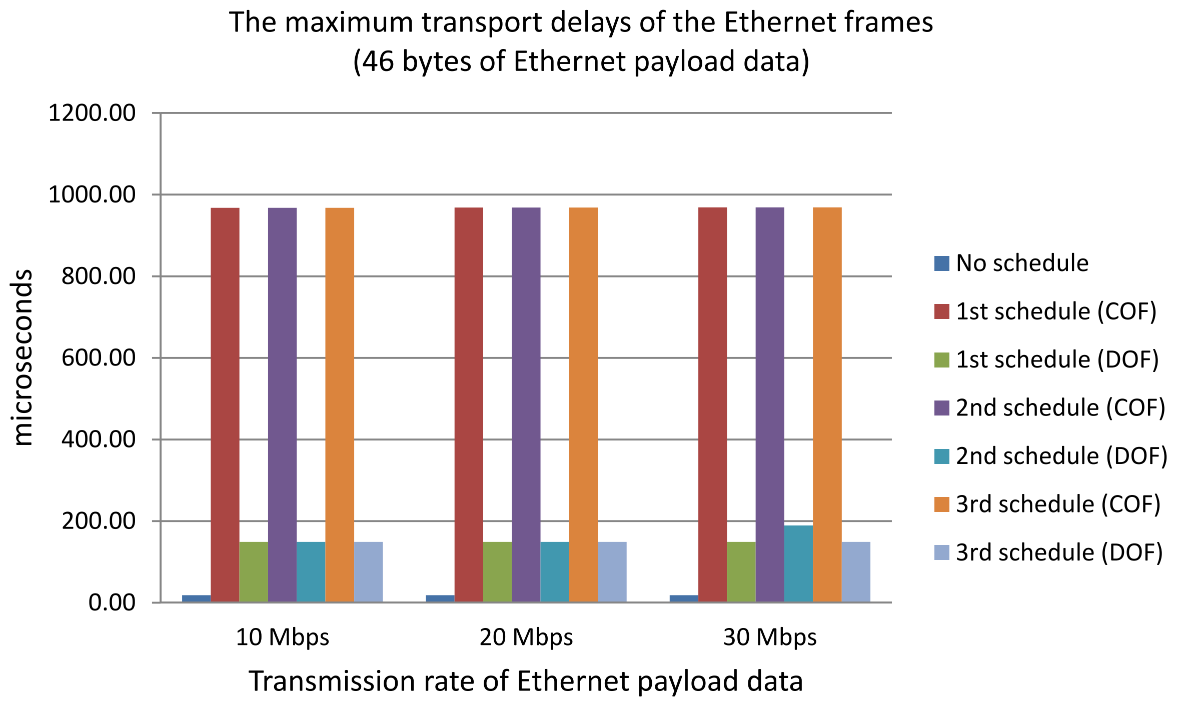


Fig. 6.6: The maximum transport delays of the standard-Ethernet traffic (46-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.6: The maximum transport delays of the standard-Ethernet traffic (46-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The maximum transport delays of the standard-Ethernet traffic with 46-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	18.39	976.76	149.04	976.76	148.84	976.76	149.00
20 Mbps	18.39	968.55	149.04	968.56	149.12	968.56	149.00
30 Mbps	18.39	968.82	149.04	968.82	189.38	968.82	149.03
40 Mbps	18.39	BOF	BOF	BOF	BOF	BOF	BOF
50 Mbps	18.39	BOF	BOF	BOF	BOF	BOF	BOF
60 Mbps	BOF	BOF	BOF	BOF	BOF	BOF	BOF
70 Mbps	BOF	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

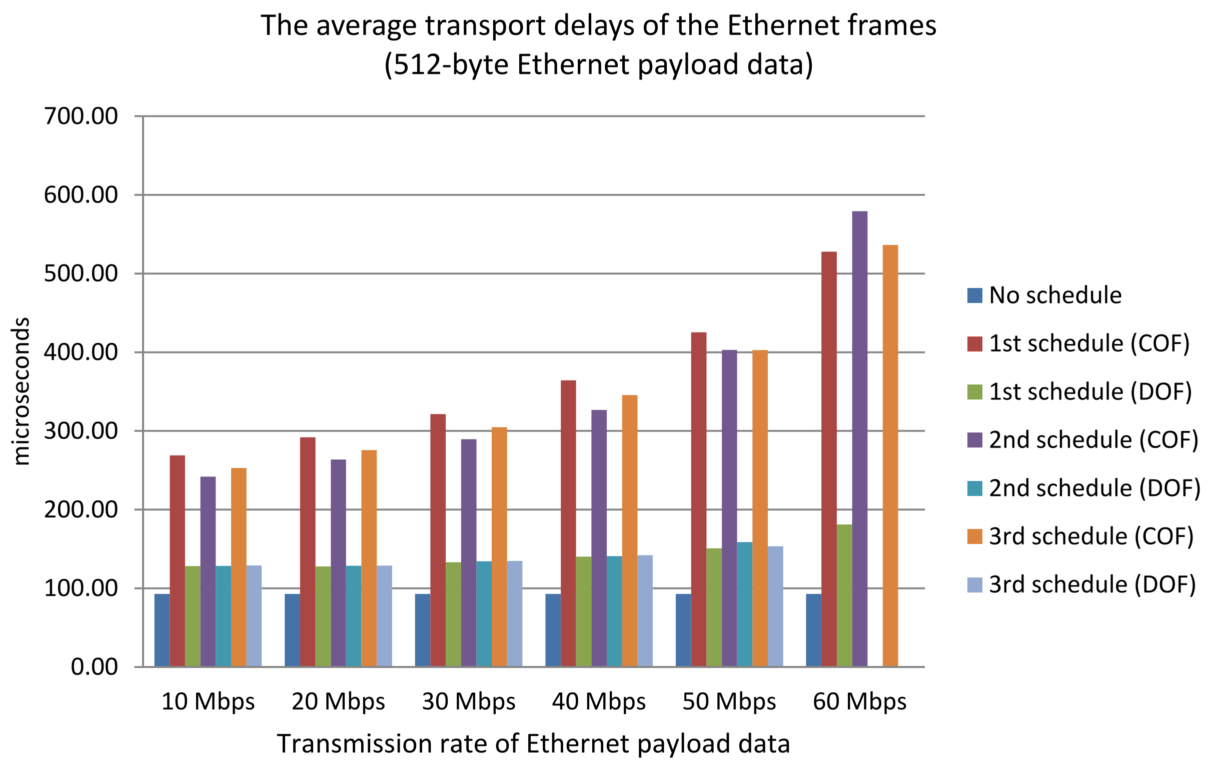


Fig. 6.7: The average transport delays of the standard-Ethernet traffic (512-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.7: The average transport delays of the standard-Ethernet traffic (512-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The average transport delays of the standard-Ethernet Ethernet traffic with 512-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	92.95	268.95	128.29	241.96	128.41	252.98	129.07
20 Mbps	92.95	291.91	127.82	263.67	128.59	275.61	128.88
30 Mbps	92.95	321.52	133.06	289.47	134.39	304.75	134.70
40 Mbps	92.95	364.30	140.27	326.68	140.86	345.66	142.19
50 Mbps	92.95	425.44	150.92	402.91	158.72	402.76	153.44
60 Mbps	92.95	527.91	181.17	579.19	BOF	536.42	BOF
70 Mbps	92.95	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

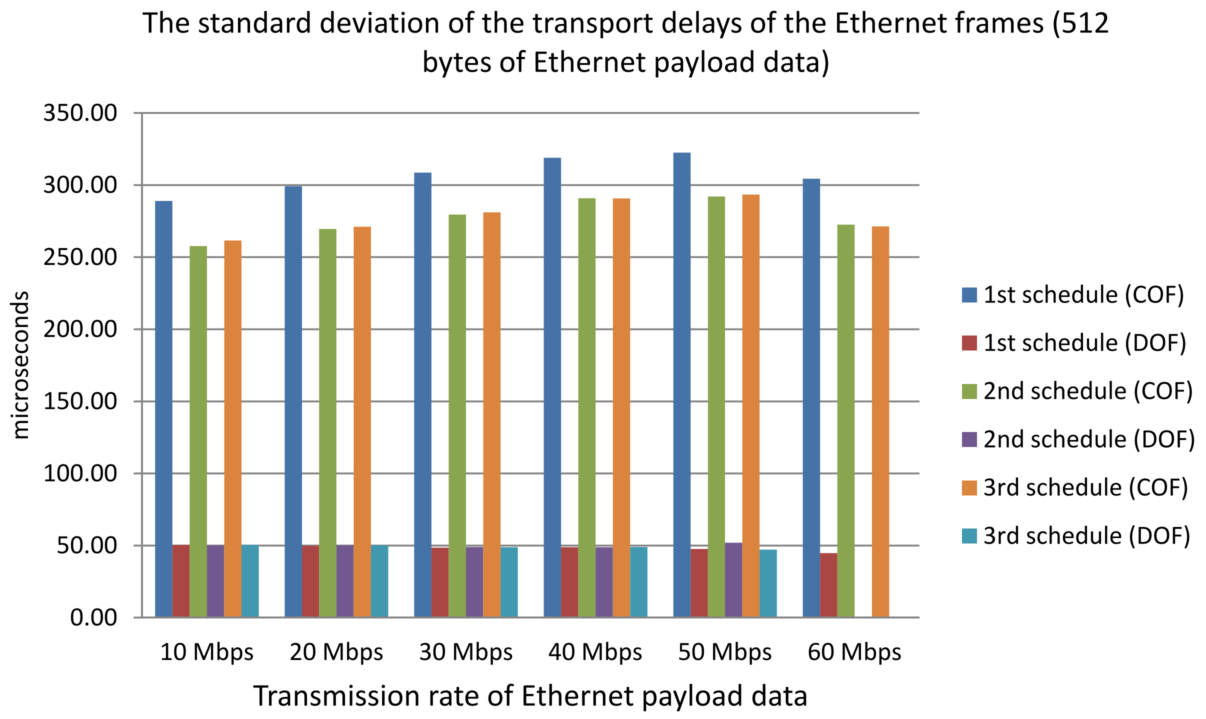


Fig. 6.8: The standard deviations of the standard-Ethernet traffic (512-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.8: The standard deviations of the transport delays of the standard-Ethernet traffic (512-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The standard deviation of the transport delays of the standard-Ethernet traffic with 512-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	0.00	288.97	50.43	257.76	50.02	261.55	50.45
20 Mbps	0.00	299.17	49.85	269.60	50.23	271.18	50.15
30 Mbps	0.00	308.62	48.56	279.59	48.93	281.10	48.88
40 Mbps	0.00	318.92	48.91	290.92	48.89	290.77	48.98
50 Mbps	0.00	322.50	47.58	292.16	51.93	293.49	47.23
60 Mbps	0.00	304.49	44.79	272.58	BOF	271.32	BOF
70 Mbps	0.00	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

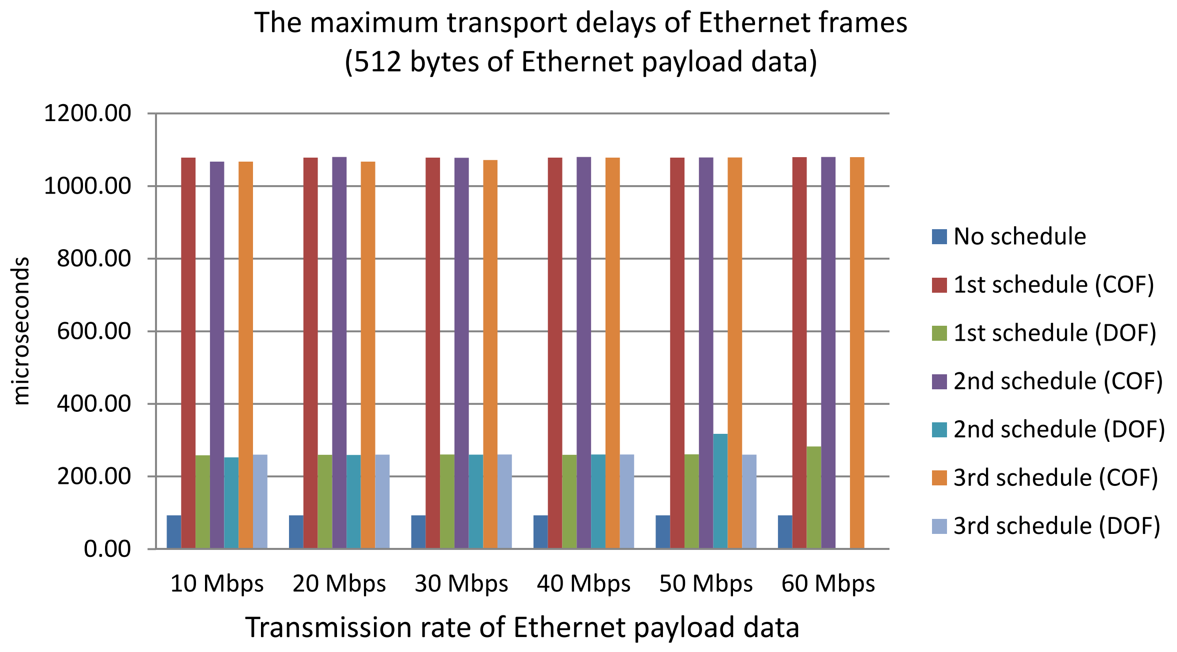


Fig. 6.9: The maximum transport delays of the standard-Ethernet traffic (512-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.9: The maximum transport delays of the standard-Ethernet traffic (512-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The maximum transport delays of the standard-Ethernet traffic with 512-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	92.95	1078.63	258.32	1067.43	252.80	1067.43	260.40
20 Mbps	92.95	1078.64	259.59	1080.24	259.20	1067.44	260.40
30 Mbps	92.95	1078.63	260.45	1078.10	260.26	1071.70	260.44
40 Mbps	92.95	1078.63	259.91	1080.24	260.63	1078.63	260.68
50 Mbps	92.95	1078.64	260.88	1078.96	317.44	1078.96	260.40
60 Mbps	92.95	1079.70	282.48	1080.24	BOF	1079.70	BOF
70 Mbps	92.95	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

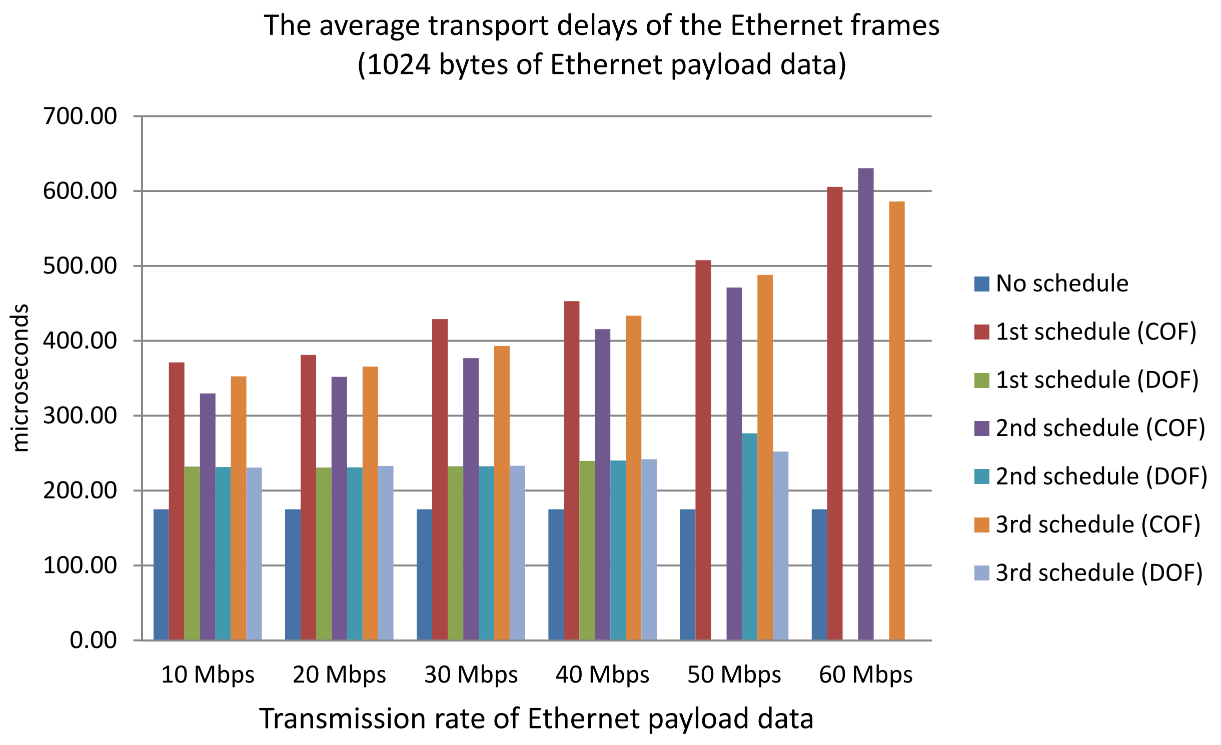


Fig. 6.10: The average transport delays of the standard-Ethernet traffic (1024-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.10: The average transport delays of the standard-Ethernet traffic (1024-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	Average transport delays of the standard-Ethernet Ethernet traffic with 1024-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	174.87	371.18	232.08	329.72	231.35	352.59	230.64
20 Mbps	174.87	381.28	230.75	351.84	230.91	365.63	232.84
30 Mbps	174.87	429.00	232.54	376.93	232.52	393.10	232.97
40 Mbps	174.87	453.01	239.40	415.60	240.11	433.53	241.80
50 Mbps	174.87	507.67	BOF	471.22	276.44	487.98	252.05
60 Mbps	174.87	605.65	BOF	630.70	BOF	586.18	BOF
70 Mbps	174.87	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

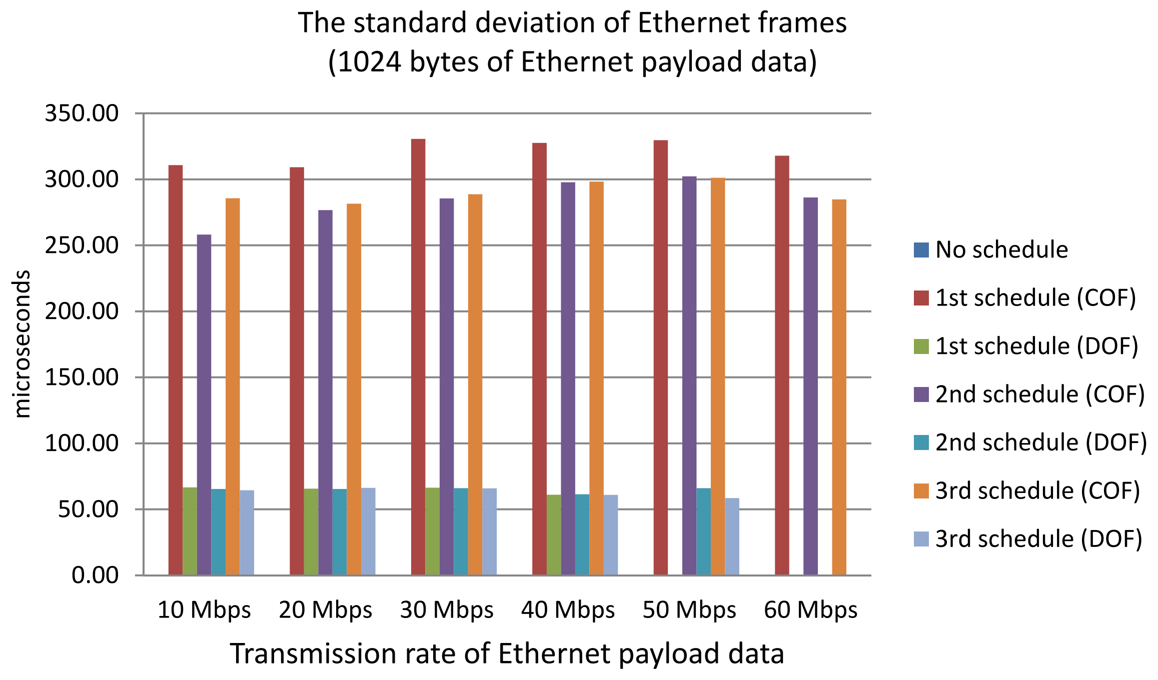


Fig. 6.11: The standard deviations of the standard-Ethernet traffic (1024-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.11: The standard deviation of the transport delays of the standard-Ethernet traffic (1024-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The standard deviation of the transport delays of the standard-Ethernet traffic with 1024-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	0.00	310.77	66.67	258.20	65.38	285.71	64.42
20 Mbps	0.00	309.16	65.65	276.73	65.45	281.60	66.29
30 Mbps	0.00	330.67	66.43	285.61	66.00	288.75	65.86
40 Mbps	0.00	327.61	61.00	297.76	61.46	298.30	60.95
50 Mbps	0.00	329.63	BOF	302.28	66.06	301.20	58.50
60 Mbps	0.00	317.98	BOF	286.33	BOF	284.79	BOF
70 Mbps	0.00	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

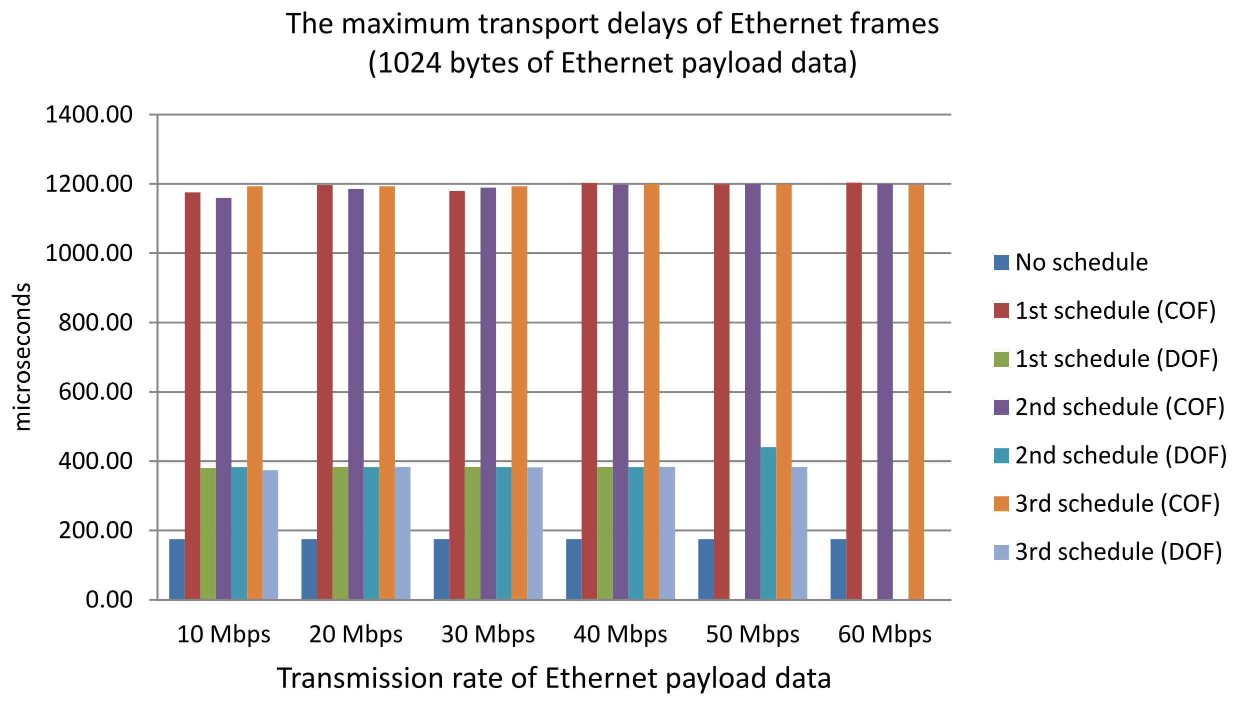


Fig. 6.12: The maximum transport delays of the standard-Ethernet traffic (1024-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.12: The maximum transport delays of the standard-Ethernet traffic (1024-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The maximum transport delays of the standard-Ethernet traffic with 1024-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	174.87	1175.60	380.72	1159.59	383.36	1193.20	373.80
20 Mbps	174.87	1196.40	383.75	1185.20	383.36	1193.19	383.23
30 Mbps	174.87	1179.33	383.54	1189.46	383.36	1193.19	381.89
40 Mbps	174.87	1202.79	383.76	1197.99	383.35	1199.60	383.24
50 Mbps	174.87	1198.96	BOF	1199.28	440.32	1198.96	383.27
60 Mbps	174.87	1203.86	BOF	1200.13	BOF	1197.46	BOF
70 Mbps	174.87	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

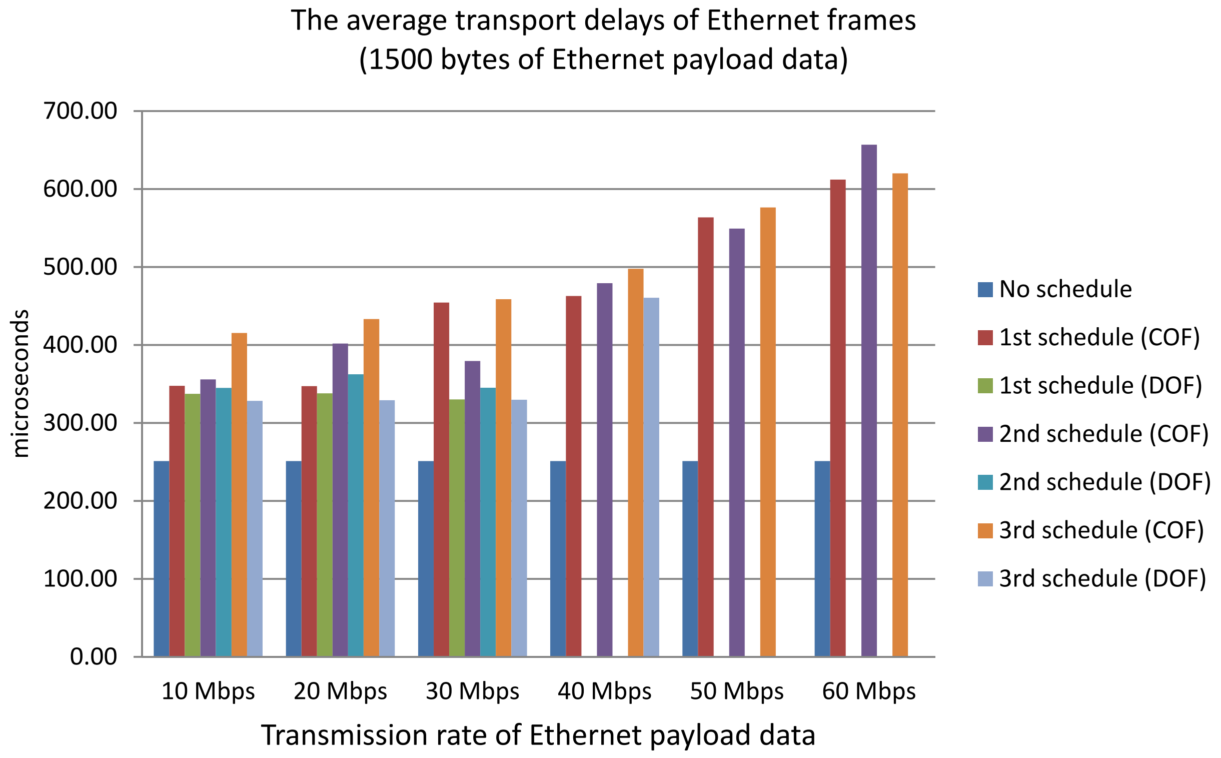


Fig. 6.13: The average transport delays of the standard-Ethernet traffic (1500-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.13: The average transport delays of the standard-Ethernet traffic (1500-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The maximum transport delays of the standard-Ethernet traffic with 1500-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	251.03	347.56	337.35	355.84	344.94	415.41	328.43
20 Mbps	251.03	347.25	338.08	401.78	362.51	433.24	329.11
30 Mbps	251.03	454.52	330.26	379.49	345.19	458.82	329.78
40 Mbps	251.03	462.82	BOF	479.37	BOF	497.86	460.54
50 Mbps	251.03	563.62	BOF	549.36	BOF	576.48	BOF
60 Mbps	251.03	612.23	BOF	656.86	BOF	620.27	BOF
70 Mbps	251.03	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

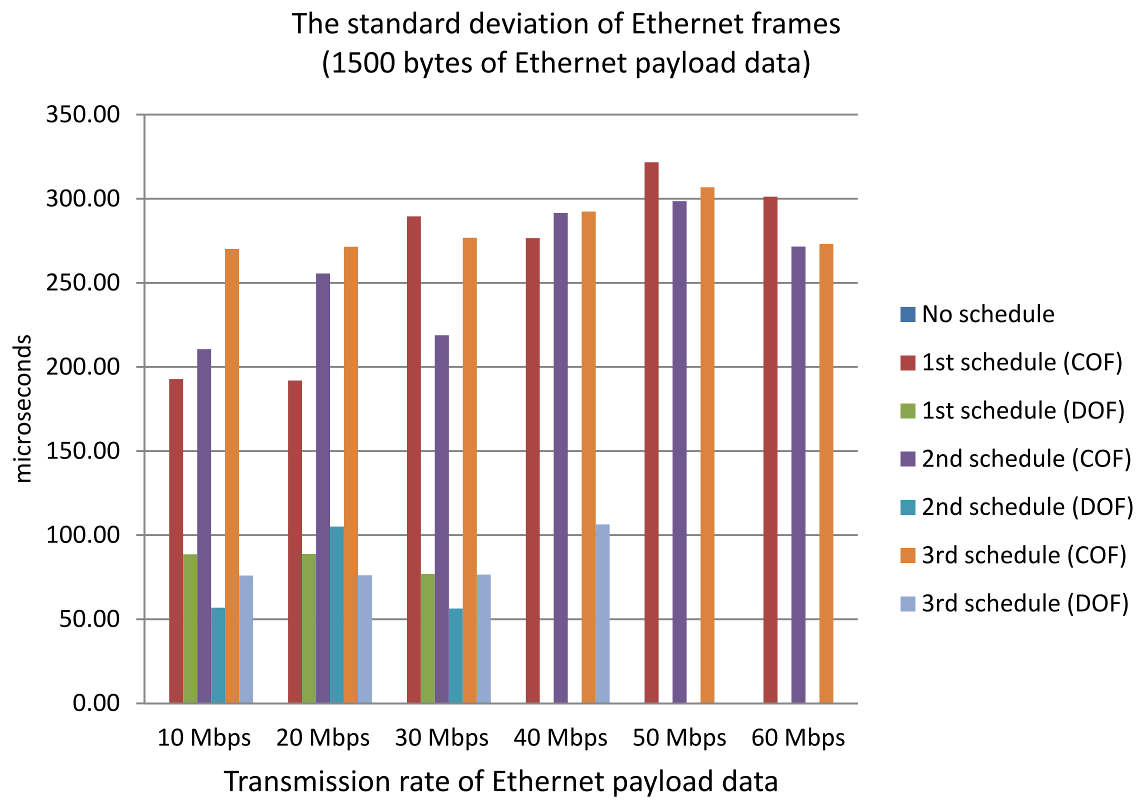


Fig. 6.14: The standard deviation of the standard-Ethernet traffic (1500-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.14: The standard deviation of the transport delays of the standard-Ethernet traffic (1500-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The standard deviation of the transport delays of the standard-Ethernet traffic with 1500-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	0.00	192.83	88.67	210.59	56.81	270.12	75.95
20 Mbps	0.00	191.89	88.81	255.59	105.12	271.42	76.17
30 Mbps	0.00	289.53	76.91	218.82	56.39	276.81	76.50
40 Mbps	0.00	276.58	BOF	291.56	BOF	292.40	106.37
50 Mbps	0.00	321.66	BOF	298.58	BOF	306.89	BOF
60 Mbps	0.00	301.24	BOF	271.59	BOF	273.07	BOF
70 Mbps	0.00	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

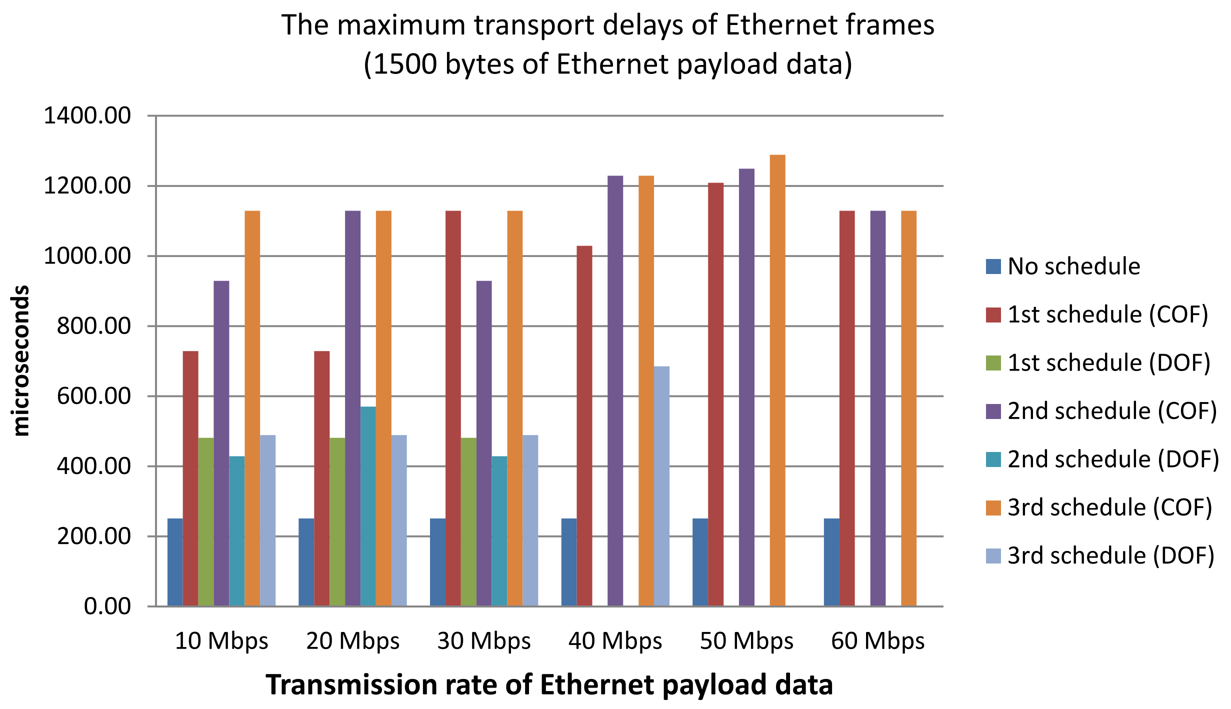


Fig. 6.15: The maximum transport delays of the standard-Ethernet traffic (1500-byte Ethernet payload data) in the simulated TTEthernet systems

Table 6.15: The maximum transport delays of the standard-Ethernet traffic (1500-byte Ethernet payload data) in the simulated TTEthernet systems

Transmission rates of Ethernet payload data	The maximum transport delays of the standard-Ethernet traffic with 1500-byte Ethernet payload data (microseconds)						
	No schedule	1st schedule		2nd schedule		3rd schedule	
		COF	DOF	COF	DOF	COF	DOF
10 Mbps	251.03	728.88	481.36	928.88	428.68	1128.88	488.96
20 Mbps	251.03	728.88	481.36	1128.88	570.24	1128.88	488.96
30 Mbps	251.03	1128.88	481.36	928.88	428.68	1128.88	488.96
40 Mbps	251.03	1028.88	BOF	1228.88	BOF	1228.88	685.12
50 Mbps	251.03	1208.88	BOF	1248.88	BOF	1288.88	BOF
60 Mbps	251.03	1128.88	BOF	1128.88	BOF	1128.88	BOF
70 Mbps	251.03	BOF	BOF	BOF	BOF	BOF	BOF

Note that COF means continuous offset form, DOF means distributed offset form, and BOF means buffer overflow

Timing-performance analysis of standard-Ethernet traffic: From the simulation results, we can analyze the average, maximum and standard deviation of the transport delays of the standard-Ethernet traffic. This is described as follows:

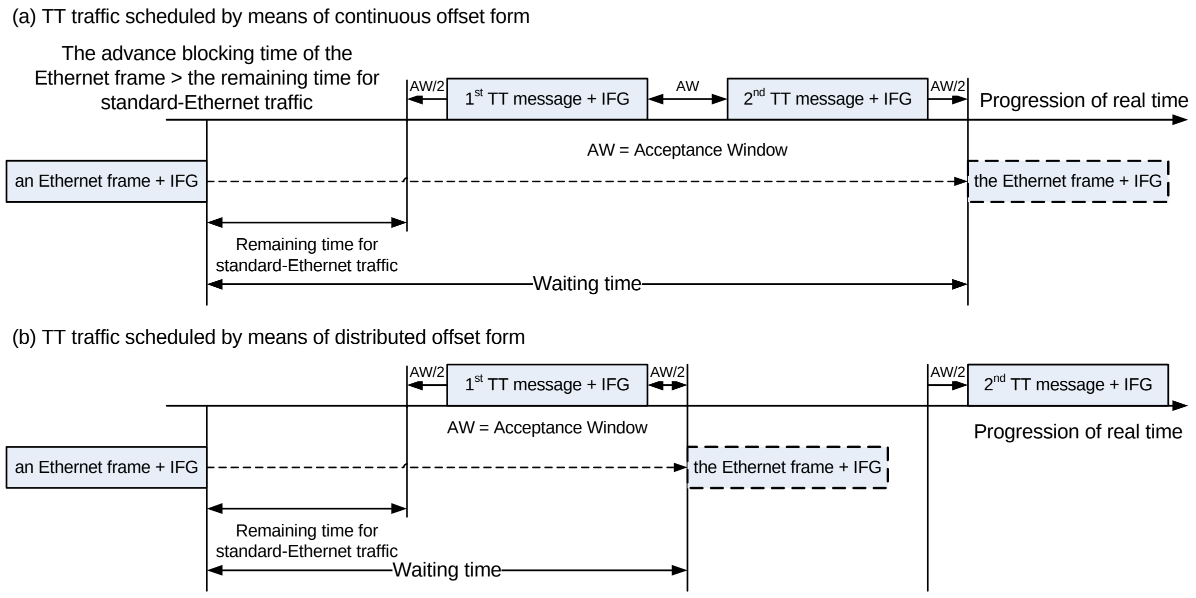


Fig. 6.16: The transmission of a standard-Ethernet frame in the simulated TTE switch in which the TT traffic is scheduled by means of (a) continuous offset form, and (b) distributed offset form

(1) The average transport delays of standard-Ethernet traffic in the simulated TTEthernet systems using schedules in distributed offset form are always smaller than those for the corresponding schedules in continuous offset form. This is because a standard-Ethernet frame has to wait in the queue buffer of the switch for its advanced blocking time plus the transmission time of TT traffic, as depicted in Fig. 6.16. In Fig. 6.16 (b), the TT traffic scheduled by means of distributed offset form makes a standard-Ethernet frame wait for the transmission of one TT message at the simulated TTE switch. This differs from the corresponding TT traffic scheduled by means of continuous offset form (Fig. 6.16 (a)), which provides a longer waiting time for the standard-Ethernet frame due to the transmission of more than one TT message.

(2) Although the schedules in distributed offset form result in high timing performance (small average transport delays) of standard-Ethernet frames, they provide less remaining bandwidth for standard-Ethernet traffic than the corresponding schedules in continuous offset form. This can be obviously seen by the buffer overflow results in Table 6.13. Note that a buffer overflow occurs when message traffic at a TTE-switch port is larger than the bandwidth. This occurred in the case of the maximum standard-Ethernet payload data size (1,500 bytes), as shown in Table 6.13. It can be seen in this table that the simulated TTE switch for standard-Ethernet traffic overflowed when the transmission rate was greater than 40 Mbps for the first and second schedule in distributed offset form, and 50 Mbps for the third schedule in distributed offset form. For the schedules in continuous offset form, the queue buffer overflowed when the transmission rate was greater than 70 Mbps. These results correspond to the standard-Ethernet throughput results in Section 6.4.1. For example, in Table 6.3, the throughputs of standard-Ethernet traffic in the case of a 1500-byte Ethernet payload data size are *36 Mbps* for the first schedule in distributed offset form. In Table 6.13, the results (average transport delays) for the system using the first schedule in distributed offset form are 330.26 microseconds, with overflow occurring when the transmission rates of Ethernet payload data are greater than *30 Mbps*.

(3) The average transport delays of standard-Ethernet frames in the simulated TTEthernet systems using schedules in distributed offset form are almost constant (or increase only slightly) with respect to higher standard-Ethernet traffic loads. This is contrary to the schedules in continuous offset form, where the average transport delays increase rapidly with higher amounts of standard-Ethernet traffic. This is because the schedules in distributed offset form have evenly distributed TT traffic in their TTE cluster cycles.

(4) The standard deviations of transport delays in systems using schedules in distributed offset form are smaller than those using schedules in continuous offset form. For example, in Table 6.11, the standard deviations of the transport delays in the case of 1024-byte Ethernet payload data and a transmission rate of 40 Mbps are 327.61, and 61.00 microseconds, for the first schedules in continuous offset form, and distributed offset form, respectively.

(5) The maximum transport delays in systems using schedules in distributed offset form are smaller than those using schedules in continuous offset form. In Fig. 6.16 (a), it is clear that a standard-Ethernet frame in continuous offset form has to wait for the transmission of more than one TT message. On the other hand, a standard-Ethernet frame in TTE systems using distributed offset form has to wait for the transmission of only one TT message, as depicted in Fig. 6.16 (b).

6.4.3 The throughputs of UDP traffic in the physical TTE systems

As mentioned earlier, we measure the throughputs of standard-Ethernet traffic in the physical TTEthernet systems using D-ITG [DITG] [BDP07]. With the D-ITG software tool, we generate standard-Ethernet UDP traffic. The Ethernet payload data size of a UDP message is equal to the summation of the IP header length (20 bytes), UDP header length (8 bytes), and the UDP message's payload data size. In the physical TTEthernet systems, TT communication schedules in continuous offset form are restricted such that the time interval between the dispatch instants of any two consecutive TT messages cannot be less than 200 microseconds. Note that this allows standard-Ethernet frames to be transmitted between any two consecutive TT messages. In this Section, we measure the maximum throughput of UDP traffic in the physical TTEthernet systems. According to Section 6.2.2, the amount of UDP traffic at the sender node for ET traffic is defined as 100% of the standard-Ethernet bandwidth. The UDP payload data size is fixed at 1,472 bytes (or 1,500 bytes of Ethernet payload data). The maximum throughputs of UDP traffic are shown in Table 6.16.

Table 6.16: The throughputs of UDP traffic (1,472 bytes of UDP payload data) in the physical TTEthernet systems

Schedules	Offset form	The UDP throughputs with 1,472 bytes of UDP payload data
No TT traffic	-	95.60 Mbit/s
The 1st schedule	Continuous offset form	65.87 MBit/s
	Distributed offset form	65.87 MBit/s
The 2nd Schedule	Continuous offset form	65.07 MBit/s
	Distributed offset form	65.07 MBit/s
The 3rd schedule	Continuous offset form	64.59 MBit/s
	Distributed offset form	64.59 MBit/s

Throughput analysis of UDP traffic: From the results shown in Table 6.16, we can analyze the UDP throughputs as follows:

(1) The UDP throughput from the physical TTEthernet system in the case of no TT traffic (95.60 Mbps) is nearly the same as the theoretical one (95.71 Mbps) derived from Equation 6.1.

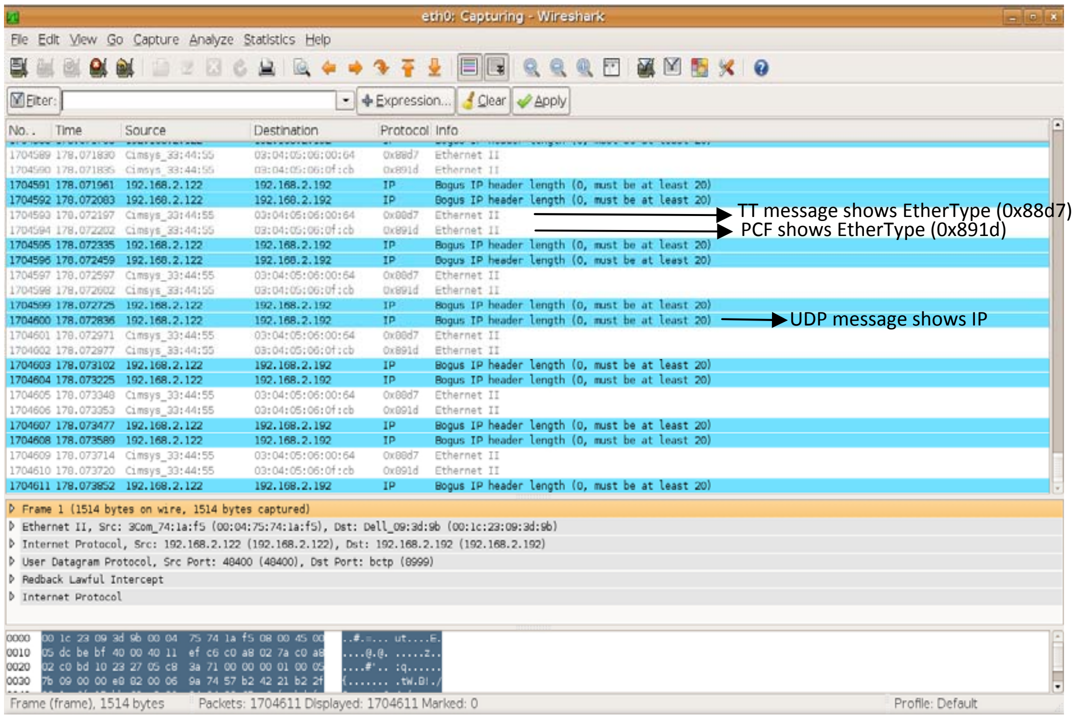
$$\text{maximum UDP throughput} = \frac{\text{UDP\_PS} \times 100\text{ Mbps}}{\text{UDP\_PS} + \text{UDP\_OL} + \text{IP\_OL} + \text{Eth\_OL}} \text{ ………………..……..(6.1)}$$

Where, UDP_PS is the UDP payload data size (1,472 bytes).

UDP_O is the UDP overhead length (20 bytes).

IP_O is the IP overhead length (8 bytes).

Eth_O is the Ethernet overhead length (Preamble + Ethernet header + Frame Check Sequence + Inter-frame gap) equal to 8 + 14 + 4 + 12 = 38 bytes.


eth0: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply
TT message shows EtherType (0x88d7)
PCF shows EtherType (0x891d)
UDP message shows IP
Profile: Default

```
No. .   Time        Source            Destination           Protocol Info
1704589 178.071830  Cimsys_33:44:55   03:04:05:06:00:64     0x88d7   Ethernet II
1704590 178.071835  Cimsys_33:44:55   03:04:05:06:0f:cb     0x891d   Ethernet II
1704591 178.071961  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)
1704592 178.072083  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)
1704593 178.072197  Cimsys_33:44:55   03:04:05:06:00:64     0x88d7   Ethernet II
1704594 178.072202  Cimsys_33:44:55   03:04:05:06:0f:cb     0x891d   Ethernet II
1704595 178.072335  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)
1704596 178.072459  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)
1704597 178.072597  Cimsys_33:44:55   03:04:05:06:00:64     0x88d7   Ethernet II
1704598 178.072602  Cimsys_33:44:55   03:04:05:06:0f:cb     0x891d   Ethernet II
1704599 178.072725  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)
1704600 178.072836  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)
1704601 178.072971  Cimsys_33:44:55   03:04:05:06:00:64     0x88d7   Ethernet II
1704602 178.072977  Cimsys_33:44:55   03:04:05:06:0f:cb     0x891d   Ethernet II
1704603 178.073102  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)
1704604 178.073225  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)
1704605 178.073348  Cimsys_33:44:55   03:04:05:06:00:64     0x88d7   Ethernet II
1704606 178.073353  Cimsys_33:44:55   03:04:05:06:0f:cb     0x891d   Ethernet II
1704607 178.073477  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)
1704608 178.073589  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)
1704609 178.073714  Cimsys_33:44:55   03:04:05:06:00:64     0x88d7   Ethernet II
1704610 178.073720  Cimsys_33:44:55   03:04:05:06:0f:cb     0x891d   Ethernet II
1704611 178.073852  192.168.2.122     192.168.2.192         IP       Bogus IP header length (0, must be at least 20)

▷ Frame 1 (1514 bytes on wire, 1514 bytes captured)
▷ Ethernet II, Src: 3Com_74:1a:f5 (00:04:75:74:1a:f5), Dst: Dell_09:3d:9b (00:1c:23:09:3d:9b)
▷ Internet Protocol, Src: 192.168.2.122 (192.168.2.122), Dst: 192.168.2.192 (192.168.2.192)
▷ User Datagram Protocol, Src Port: 48400 (48400), Dst Port: bctp (8999)
▷ Redback Lawful Intercept
▷ Internet Protocol

0000  00 1c 23 09 3d 9b 00 04  75 74 1a f5 08 00 45 00   ..#.=... ut....E.
0010  05 dc be bf 40 00 40 11  ef c6 c0 a8 02 7a c0 a8   ....@.@. .....z..
0020  02 c0 bd 10 23 27 05 c8  3a 71 00 00 00 01 00 05   ....#'.. :q......
0030  7b 09 00 00 e8 02 00 06  9a 74 57 b2 42 21 b2 2f   {....... .tW.B!./

Frame (frame), 1514 bytes    Packets: 1704611 Displayed: 1704611 Marked: 0
```

Fig. 6.17: The messages transmitted from the physical TTE switch having a TT-communication schedule (with a period and offset of 400 and 200 microseconds, respectively).

(2) The UDP throughputs in the physical TTEthernet network using schedules in continuous offset form are equal to those in the corresponding schedules in distributed offset form. For example, UDP throughput using the first schedule in continuous offset form (65.87 Mbps) is equal to that of the first schedule in distributed offset form. We have found that the mechanism in the physical TTE switch is different from the one used in our simulation model. The physical switch uses the *shuffling* method [SBH09] to handle conflicting standard-Ethernet and TT traffic. This denotes that if a TT message requires transmitting at a port during transmission of a standard-Ethernet frame, the standard-Ethernet transmission continues regardless. The TT message is therefore transmitted after the transmission of the standard-Ethernet frame. To prove that the physical TTE switch supports the *shuffling* method, we setup a TT communication schedule consisting of one time-critical application (TT traffic with a transmission period of 400 milliseconds), and PCF traffic with a transmission period of 400 milliseconds. Each of the TT messages of the time-critical application has 1,500 bytes of Ethernet payload data. The time-critical application is scheduled with an offset of 200 milliseconds. With this schedule, a simulated TTE switch allows transmission of one standard-Ethernet frame after the transmission of a PCF message, but before the transmission of the consecutive TT

message. The result showed that there is more than one standard-Ethernet frame transmitted at the switch, as depicted in Fig. 6.17. Therefore, the bandwidth utilization for standard-Ethernet traffic in the physical TTE switch is high, even though the TT traffic is scheduled in distributed offset form.

(3) The UDP throughputs in the physical TTE network are higher than those in the simulated network scheduled in distributed offset form. In our simulation model, we use the *timely block* method [SBH09] to handle conflicting standard-Ethernet and TT traffic. The *timely block* method guarantees that all standard-Ethernet frames in a TTE switch have completely finished transmitting before the forthcoming dispatch instant of TT traffic. With this method, the timing performance of TT traffic is maximized (i.e. there is no interference from the transmission of standard-Ethernet traffic to the timing performance of TT traffic). However, the *timely block* method causes standard-Ethernet frames to be blocked in the queue buffer of a TTE switch for their advanced blocking time, plus the forthcoming TT-message transmission time. This is explained in Section 6.4.1. With regard to TT traffic scheduled in distributed offset form, the bandwidth utilization for standard-Ethernet traffic in the physical TTE switch (using the *shuffling* method) is higher than that in the simulated TTE switch (using the *timely block* method). This is because there is no advanced blocking time for TT traffic in the physical TTE switch.

(4) The result of UDP throughput in the physical TTEthernet systems in each schedule (shown in Table 6.16) correspond to the remaining bandwidths for standard-Ethernet traffic, as shown in Table 6.2. This table shows the first schedule giving the highest UDP throughput (65.87 Mbps) of all schedules in the physical network. Similarly, the first schedule also provides the highest remaining bandwidth for standard-Ethernet traffic (69,024 kbps). In the physical TTEthernet systems, the UDP throughput of the third schedule is the lowest (64.59 Mbps). Similarly, the third schedule has the lowest remaining bandwidth for standard-Ethernet traffic (67,686.4 kbps). Both these results correspond to each other, because the physical TTE switch employs the *shuffling* method (i.e. there is no advanced blocking time). Therefore, the TT communication schedule with the highest remaining bandwidth for standard-Ethernet traffic must also provide the highest throughput of Ethernet-based messages.

6.4.4 Round trip delays of standard-Ethernet traffic in the physical TTEthernet systems

We use the ICMP request/reply known as the 'ping' command for measuring round trip delays of standard-Ethernet traffic. An ICMP message is encapsulated by the Internet Protocol (IP) and standard Ethernet. The Ethernet payload data size of an ICMP message is equal to the summation of the IP header length (20 bytes), the ICMP header length (8 bytes), and the ICMP message's payload data size. In Fig. 6.1, the sender node for TT traffic is responsible for generating ICMP messages. The ICMP traffic is defined according to Section 6.2.2.2, where the ICMP payload data size is 1,472 bytes (or 1,500 bytes of Ethernet payload data). The inter-departure time between any two ICMP messages is 200 microseconds. The measurement results show the minimum, average and standard deviation of the round trip delays for all transported ICMP messages. This is shown in Table 6.17.

Table 6.17: The minimum, average and standard deviation of the round trip delays of the ICMP traffic (1,472 bytes of ICMP payload data) in the physical TTEthernet systems

Schedules	Offset forms	Round trip delays of ICMP traffic (us)		
		Minimum round trip delay	Average round trip delays	Standard deviation
First schedule	Continuous offset form	626.0	735.6	44.2
	Distributed offset form	626.0	728.4	38.5
Second Schedule	Continuous offset form	626.0	755.4	50.9
	Distributed offset form	626.0	742.4	50.6
Third schedule	Continuous offset form	626.0	738.0	41.3
	Distributed offset form	626.0	726.3	38.7

Timing-performance analysis of ICMP traffic: From the results of the round trip delays in the physical TTEthernet network, we can analyze the minimum, average and standard deviation of the round trip delays of all transported ICMP traffic. This is described as follows:

(1) The average round trip delays of ICMP traffic in the physical TTEthernet systems using schedules in distributed offset form are smaller than those for the corresponding schedules in continuous offset form. For example, the average round trip delays using the first schedules in both continuous offset form and distributed offset form are 735.6 and 728.4 microseconds, respectively. Note that the schedules in continuous offset form (with the constraint described in Section 6.3) are analogous to schedules in distributed offset form where the TT traffic is unevenly distributed in the TTE cluster cycle. However, in our scheduling approach, TT traffic scheduled in distributed offset form is evenly distributed in the TTE cluster cycle. The average round trip delays of Ethernet-based messages in the physical TTEthernet systems (shown in Table 6.17) correspond to the average transport delays of standard-Ethernet frames in the simulated TTEthernet systems. This means that schedules in distributed offset form provide lower average transport delays in the simulated TTEthernet network than schedules in continuous offset form.

(2) The standard deviations of the ICMP round trip delays in the physical TTE systems using schedules in distributed offset form are smaller than the corresponding ones in continuous offset form. As shown in Table 6.17, the standard deviations for the first schedules in continuous offset form and distributed offset form are 44.2 and 38.5 microseconds, respectively. This means that a schedule where TT traffic is *evenly* distributed results in smaller standard deviations than the corresponding schedule where TT traffic is *unevenly* distributed (a schedule in continuous offset form). This corresponds to the results in the simulated TTEthernet systems. Thus, schedules in distributed offset form always result in smaller ICMP transport delay standard deviations than the corresponding schedules in continuous offset form.

(3) The timing performance of Ethernet-based messages in the physical TTEthernet network depends on the arrangement of TT traffic in the TT communication schedule. As shown in Table 6.17, the schedules in distributed offset form provide better timing performance of ICMP traffic than the corresponding schedules in continuous offset form.

(4) The minimum round trip delay of ICMP traffic is the best case (shortest) round trip delay. This occurs when an ICMP message is transported into a TTEthernet network without any interference from TT traffic. As shown in Table 6.17, the minimum round trip delay of ICMP traffic in all schedules is equal.

6.5 Summary

In this chapter we have analyzed the performance of standard-Ethernet traffic in both the simulated and physical TTEthernet systems. In the simulated TTEthernet systems based on our simulation model, the first schedule in continuous offset form resulted in the highest standard-Ethernet throughput. The standard-Ethernet throughputs from the simulated TTEthernet systems in distributed offset form were lower than the corresponding ones in continuous offset form. In the simulated TTEthernet systems, the TTE switch utilized the *timely block* method to handle conflicting TT and standard-Ethernet traffic. By doing this it obtained the maximum performance of TT traffic, due to the standard-Ethernet traffic not being able to interfere with the timely behavior of the TT traffic. In the physical 100-Mbps TTEthernet systems, the first schedule in continuous offset form resulted in the highest throughput of UDP traffic. On the contrary to the simulated TTEthernet systems, the physical TTE systems had equal throughput performance for both distributed offset form and continuous offset form. This is because of the following two reasons. Firstly, the physical TTE switch (which employs the *shuffling* method to handle conflicting TT and standard-Ethernet traffic) allows a standard-Ethernet frame to continue its transmission until completion, even though the dispatch instant of a TT message may have been reached. The second reason is that in the physical TTEthernet systems, TT traffic scheduled in continuous offset form has to be scheduled with a constraint so that the time interval between any two consecutive TT messages is not less than 200 microseconds. Therefore, standard-Ethernet frames in the physical TTE switch using continuous offset form can be transmitted during such a time interval. With regard to the timing performance of standard-Ethernet traffic, TT traffic scheduled in distributed offset form resulted in better timing performance of standard-Ethernet traffic than that TT traffic scheduled in continuous offset form.

CHAPTER 7

Conclusion

In this chapter, we present the conclusion of this thesis. According to our literature review we found that most protocols of *real-time Ethernet* were not aimed to altogether achieve the three main properties: (1) determinism property for real-time traffic, (2) full standard-Ethernet compatibility for real-time communication, and (3) high Ethernet-bandwidth utilization for coexisting standard-Ethernet traffic. For TTEthernet these three properties were focused in TTEthernet's design. With the time-triggered communication paradigm [Kop97] TTEthernet fully support the determinism property [KG93], which is used for developing highly safety-critical systems. Regarding the second property, TTEthernet was designed to expand standard Ethernet [IEE05] with services to meet time-critical, deterministic, and safety-relevant conditions [TTE08]. Therefore there is no modification of standard-Ethernet hardware for TTEthernet implementation. In the last property TTEthernet achieve the high Ethernet-bandwidth utilization for coexisting standard-Ethernet traffic. This is because the coexisting standard-Ethernet traffic can flow through TTE switches regardless of whether they are in the synchronization state of TTEthernet. However the flow of standard-Ethernet traffic in TTE switches which are in the synchronization state of TTEthernet depends on their common TT-communication schedule. This is due to the fact that the TT traffic takes precedence over standard-Ethernet traffic. In this thesis we investigate a TT-traffic scheduling approach for obtaining the high performance of standard-Ethernet traffic coexisting with the TT traffic in the same TTEthernet network. This thesis contains three main contributions: the simulation model for TTEthernet systems, the TT-traffic scheduling approach, and the performance evaluation of standard-Ethernet traffic in TTEthernet systems. We conclude our work results with regard to three contributions as follows.

7.1 The simulation model for TTEthernet systems

We have proposed the simulation model for TTEthernet systems. The design of our simulation model is based on the concept of discrete event simulation. With this simulation model, all the components of TTEthernet systems were described. We have implemented the simulation from our simulation model, using a Java runtime environment and the Java library "J-Sim". Due to the determinism property of TTEthernet for TT traffic, we validate the flow of the TT communication of simulated TTEthernet systems with a predefined confliction-free TT communication schedule. Since the communication of standard Ethernet is non-deterministic, we have calibrated the simulation results from our simulation model with the analytical ones from the published papers [LL02] [Rug08]. The calibration outcome shows that the results from the simulations based on our simulation model coincide with the analytical ones.

7.2 The TT-traffic Scheduling approach for TTEthernet systems

We have proposed the TT-traffic scheduling approach for TTEthernet systems. Our TT-traffic scheduling approach aims to find an optimal TT-communication schedule for standard-Ethernet traffic performance in TTEthernet systems. In this thesis the throughput and timing-performance of standard-Ethernet traffic in TTEthernet systems are the key performance parameters. By using our TT-traffic scheduling approach we can enumerate various TT-communication schedules (new periods) from the specified periods of given time-critical applications. With each TT-communication schedule the amount of remaining bandwidth for standard-Ethernet traffic is then analyzed mathematically. The TT-communication schedule resulting in the highest amount of remaining bandwidth for standard-Ethernet traffic is supposed to be the optimal one (optimal periods). The length of a TT-communication schedule denotes the TTE cluster cycle length. It is calculated by means of Least Common Multiple (LCM). In term of offset for TT-communication schedules, we classified offset forms into two possible forms: *continuous offset form*, and *distributed offset form*. With both offset forms, the analysis of the maximum waiting time of standard-Ethernet frames due to TT-message transmission was presented. We found that a TT-communication schedule in distributed offset form results less maximum waiting time of standard-Ethernet frames than that in continuous offset form. Furthermore we presented the methodology applying the obtained periods and offset to the dispatching and receiving points in time for all TT messages and PCFs in the TTE cluster cycle. A TTE cluster cycle along with dispatching and receiving points in time for TT message and PCFs are the TT-communication schedules. Based on our analysis, the TT-communication schedule with the highest remaining bandwidth for standard-Ethernet traffic, and in distributed offset form is supposed to be the optimal one.

7.3 Performance Analysis of standard-Ethernet traffic in TTEthernet systems

We have analyzed the performance of standard-Ethernet traffic in both simulated and physical 100-Mbps TTEthernet systems. The key performance parameters of standard-Ethernet traffic are throughout and latency. In the simulated TTEthernet systems, the simulated TTE switch uses the *timely block* method to handle conflicting time-triggered (TT) and standard-Ethernet traffic. With the *timely block* method, the performance of TT traffic is maximized (i.e. there is no interference from standard-Ethernet traffic to the TT traffic). With regard to the throughput of standard-Ethernet traffic, the simulated TTEthernet systems using a TT-communication schedule in continuous offset form provide better throughput of standard-Ethernet traffic than those using the corresponding schedule in distributed offset form. This is because a TT-communication schedule in distributed offset form results in a higher amount of *advanced blocking time for TT traffic* than the corresponding one in on continuous offset form. An *advanced blocking time for TT traffic* is the time interval which a standard-Ethernet frame is blocked in the queue buffer of a TTE switch before the forthcoming TT-message transmission in order to avoid the message confliction. With regard to the timing performance of standard-Ethernet traffic, the simulation outcome shows that the simulated TTEthernet systems using a TT-communication schedule in distributed offset form gives better timing-performance of standard-Ethernet traffic than those using the corresponding schedule in continuous offset form. Notice that there is a tradeoff between the throughput and timing-performance of standard-Ethernet traffic in the simulated TTEthernet systems using the TT-communication schedules in continuous offset form and distributed offset form.

In the physical 100-Mbps TTEthernet systems, the TTE switch uses the *shuffling* method to handle conflicting TT and standard-Ethernet traffic. The experimental results show that a schedule in continuous offset form results in the same UDP throughput performance as that in distributed offset form. This is because the *shuffling* method does not cause advanced blocking time for TT traffic. In additional, the experimental results of UDP throughput in the physical TTEthernet systems correspond to the remaining bandwidths for standard-Ethernet traffic calculated using our TT-scheduling approach. The first schedule results in both the highest UDP throughput in the physical TTEthernet system, and the highest remaining bandwidth for standard-Ethernet traffic. With regard to the timing-performance of standard-Ethernet traffic, the round trip delays of ICMP traffic in the physical TTEthernet systems were observed. The results show that TT traffic scheduled in distributed offset form gives better timing-performance of standard-Ethernet traffic than that scheduled in continuous offset form. In the physical TTEthernet systems, TT-communication schedules based on continuous offset form are restricted such that the time interval between the dispatch instants of any two consecutive TT messages cannot be less than 200 microseconds. Note that this allows standard-Ethernet frames to be transmitted between any two consecutive TT messages. In the end we conclude that the first schedule in distributed offset form is an optimal one for physical 100-Mbps TTEthernet systems.

7.4 Outlook

In the future, our research work focuses on the performance enhancement of standard-Ethernet traffic in TTE switches in which the *timely block* method is used. With the *timely block* method, the performance of TT traffic in TTEthernet systems is maximized (i.e. there is no interference from standard-Ethernet traffic to the TT traffic). According to the results (both throughput and transport delay) in the simulated TTEthernet systems, there is a tradeoff between the throughput and timing-performance of the standard-Ethernet traffic. This tradeoff should be solved in order to gain higher performance of standard-Ethernet traffic in TTEthernet systems. In the future, therefore we tend to enhance throughput performance for standard-Ethernet traffic in TTE switches during advanced blocking time for TT traffic. With such that, TT traffic scheduled in distributed offset form can achieve both high throughput and timing-performance for standard-Ethernet traffic in TTEthernet systems.

Appendix A
List of Abbreviations

ACCO	Active Control Connection Object
ADU	Application Data Unit
AFDX	Avionics Full Duplex Switch Ethernet
AR	Application Relationship
ARINC	Aeronautical Radio, Incorporated
ASIC	Application-Specific Integrated Circuit
AT	Answer Telegram
BAG	Bandwidth Allocation Gap
BC	Best-Effort
BOF	Buffer Overflow
CBA	Component Based Automation
CC	Cross Communication
CI	ControlNet International
CIP	Common Industrial Protocol
CN	Controller Node
COF	Continuous Offset Form
COM	Component Object Model
COTS	Commercial Off-The-Shelf
CR	Communication Relationship
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access/Collision Detection

C2C	Controller to Controller
DCOM	Distributed Component Object Model
DOF	Distributed Offset Form
D-ITG	Distributed Internet Traffic Generator
EANTC	European Advanced Networking Test Center
EPLGW	Ethernet to Ethernet POWERLINK Gateway
EPSG	Ethernet POWERLINK Standardization
ET	Ethernet
FIFO	First-In First-Out
FPGA	Field Programmable Gate Array
GSD	General Station Description
FCS	Frame Check Sequence
LAN	Local Area Network
LCM	Least Common Multiple
LD	Logical Device
Lmax	Largest Ethernet Frame
MAC	Media Access Control
ICMP	Internet Control Message Protocol
IEA	Industrial Ethernet Association
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IFG	Inter-frame Gap
IP	Internet Protocol
IPG	Inter-packet Gap
IPv4	Internet Protocol version 4

IRT	Isochronous Real-Time
I/O	Input/Output
MBAP	MODBUS Application Protocol
MDT	Master Data Telegram
MN	Managing Node
MST	Master Synchronization Telegram
NRT	Non Real-Time
ODVA	Open DeviceNet Vendors Association
OSI	Open System Interconnection
PCF	Protocol Control Frame
PD	Physical Device
PDU	Protocol Data Unit
PI	PROFIBUS International
PLC	Programmable Logic Controller
PReq	Poll Request
PRes	Poll Response
PTCP	Precision Transport Clock Protocol
QoS	Quality of Service
RBSE	Remaining Bandwidth for Standard-Ethernet Traffic
RC	Rate-Constrained
RT	Real-Time
RT-Auto	Runtime Auto
SAP	Service Access Point
SDO	Service Data Object
SoA	Start of Asynchronous

SoC	Start of Cycle
SN	Sequence Number
PDO	Process Data Object
TAI	International Atomic Time
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TDMA	Time Division Multiple Access
TFTP	Trivial File Transfer Protocol
TTP	Time-Triggered Protocol
TT	Time-Triggered
TTE	TTEthernet
TTL	Time-To-Live
UBTT	Used Bandwidth for Time-Triggered Traffic
UDP	User Datagram Protocol
VL	Virtual Link
VLAN	Virtual Local Area Network

Bibliography

[Abr77] Norman Abramson. The Throughput of Packet broadcasting channels, IEEE Transactions on Communications, Vol. 25, No. 1, pp. 117-128, January 1977.

[Acr05] ACROMAG Incorporated. Introduction to MODBUS TCP/IP. http://www.acromag.com/sites/default/files/Acromag_Intro_ModbusTCP_765A.pdf, 2005. Accessed 8-January-2011.

[Act05] Actel Corporation. Application Note AC221: Developing AFDX Solutions. http://www.actel.com/documents/AFDX_Solutions_AN.pdf, March 2005. Accessed 10-March-2011.

[All09] Allied Telesis Inc. Datasheet of AT-FS750/16 16 port Fast Ethernet WebSmart Switch. http://www.alliedtelesis.com/media/datasheets/fs750-16_ds.pdf, 2009. Accessed 20-November-2010.

[Ana02] Moldeovansky Anatoly. Utilization of Modern Switching Technology in EtherNet/IP Networks. In *Proceedings 1st International Workshop on Real-Time LANs in the Internet Age* (RTLIA 2002) in conjuction with the 4th Euromicro Conference on Real-Time Systems, Vienna, Austria, June 18, 2002.

[ARI04] ARINC. 429 Mark 33 Digital Information Transfer System (DITS), May 2004. Part 1: Functional Description, Electrical Interface, Label Assignments and Word Formats, Part 2: Discrete Data Standards, Part 3: File Data Transfer Techniques.

[ARI05] ARINC Specification 653, Dec 2005. Avionics Application Software Standard Interface.

[ARI06] ARINC. 664 ARICRAFT DATA NETWORK, June 2006. Part 1: Systems Concepts and Overview, Part 2: Ethernet Physical and Data Link Layer Specification, Part 3: Internet-based Protocols and Services, Part 4: Internet-based Address Structures and Assigned Numbers, Part 5: Network Domain Characteristics and Interconnection, Part 6: Reserved, Part 7: Avionics Full Duplex Switched Ethernet (AFDX), Part 8: Interoperation with Non-IP Protocols and Services.

[ARI99] ARINC. 629 Multi-Transmitter Data Bus, March 1999. Part 1: Technical Description, Part 2: Application Guide.

[AVI11] Avionics Interface Technologies Company. White Paper: SAE AS6802 DETERMINISTIC ETHERNET NETWORK SOLUTION, http://www.aviftech.com, March 2011. Accessed 18-May-2011.

[BCF98] Manuel B. M. Barbosa, Adriano da Silva Carvalho, Mohammed Farsi. A CANopen I/O Module: Simple and Efficient System Integration, In *Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society*, Vol. 1, pages 155-159, August 31 – September 4, 1998, Aachen, Germany.

[BDP07] Alessio Botta, Alberto Dainotti, Antonio Pescapè, "Multi-protocol and multi-platform traffic generation and measurement", INFOCOM 2007 DEMO Session, May 2007, Anchorage (Alaska, USA).

[BGM10] R K Bansal, Vikas, Gupta, Rahul Malhotra. Performance Analysis of Wired and Wireless LAN Using Soft Computing Techniques-A Review, in *Global Journal of Computer Science and Technology*, Vol.10, Issue 8, September 2010, pp. 67-71.

[BOX98] D. Box. *Essential COM*, AddisonWesley Publishing Company, 1998.

[BSK11] Florian Bartols, Till Steinbach, Franz Korf, Thomas C. Schmidt. Performance Analysis of Time-Triggered Ether-Networks Using Off-The-Shelf-Components, In *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2011)*, March 28-31, 2011, Newport Beach, California, USA.

[Cis09] Cisco Systems Inc. Datasheet - Cisco Catalyst 3560 Series Switches. http://www.cisco.com/en/US/hmpgs/index.html, 2009. Accessed 14-May-2011.

[Con05] Condor Engineering. AFDX/ARINC 664 Tutorial. http://www.cems.uwe.ac.uk/~ngunton/afdx_detailed.pdf, May 2005. Accessed 10-March-2011.

[Con10] ControlDesign. Three Variants Dominate Industrial Ethernet. http://www.controldesign.com/industrynews/2010/021.html, April 29, 2010. Accessed 19-November-2010.

[CSCM06] Vicente Casanova, Julián J. Salt, Ángel Cuenca and Vicente Mascarós. Networked Control Systems over Profibus-DP: Simulation Model, In *Proceedings of the 2006 IEEE International Conference on Control Applications*, Munich, Germany, October 4-6, 2006.

[CSV08] G. Cena, L. Seno, A. Valenzano, S. Vitturi. Performance Analysis of Ethernet Powerlink networks for distributed control and automation systems. *Computer Standard & Interfaces*, Vol. 31, pages 566-572, 2009.

[CXW09] Xin Chen, Xudong Xiang, Jianxiong Wan. A Software Implementation of AFDX End System, In *Proceedings of the 2009 International Conference on New Trends in Information and Service Science*, Beijing, China, June 30 – July 2, 2009.

[Dec05] Jean-Dominique Decotignie. Ethernet-Based Real-Time and Industrial Communications, *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1102-1117, June 2005.

[DITG] D-ITG (Distributed Internet Traffic Generator), http://www.grid.unina.it/ software/ITG, Accessed 20-June-2011.

[Doy04] Paula Doyle. Introduction to Real-Time Ethernet I. in *the EXTENSION: A Technical Supplement to Control Network,* Vol. 5, Issue 3, May-June, 2004.

[DXL10] Song Dong, Zeng Xingxing, Ding Lina. The Design and Implementation of the AFDX Network Simulation System, *The 2010 International Conference on Multimedia (ICMT)*, Ningbo, China, Oct. 29-31, 2010.

[EAN09] EANTIC Test Report. *Arista 7148SX Unicast and Multicast Performance and Latency Tests*. http://www.eantc.com/fileadmin/eantc/downloads/ test_reports/2009-2011/EANTC-Arista-Marketing_Report-1.3.pdf, 2009. Accessed 04-May-2011.

[EPS08] EPSG – Ethernet POWERLINK Standardization Group. Ethernet POWERLINK Basics. http://www.ethernet-powerlink.org, 2008. Accessed 19-January-2011.

[Eth09] EtherCAT Technology Group. EtherCAT Introduction. http://www.ethercat.org/pdf/english/EtherCAT_Introduction_0905.pdf, September 2009. Accessed 15-January-2011.

[Eth10a] EtherCAT Technology Group. Industrial Ethernet Technology Comparison. http://www.ethercat.org/en/publications.html, Nueremburg, August 2010. Accessed 3-January-2011.

[Eth10b] EtherCAT Technology Group. EtherCAT – Technical Introduction and Overview. http://www.ethercat.org/pdf/english/ETG_Brochure_EN.pdf, September 2010. Accessed 15-January-2011.

[Fei00] Sidnie Feit. Local Area High Speed Networks. MacMillan Technical Publishing, First edition, June 2000. ISBN 1-57870-113-9.

[Fel04] Joachim Feld. PROFINET - Scalable Factory Communication for all Applications. In *2004 IEEE International Workshop on Factory Communication Systems*, pages 33-38, Sep. 22-24, 2004, Vienna, Austria, 2004.

[Fel05] Max Felser. Real-Time Ethernet – Industrial Perspective. *Proceedings of the IEEE,* vol. 93, no. 6, pp. 1118 – 1129, June 2005.

[FFV06] P. Ferrari, A. Flammini, S. Vitturi. Performance Analysis of PROFINET Networks. *Computer Standards and Interfaces, Volume: 28, Number: 4,* pages 369-385, April 2006.

[Flu08] Fluke Networks. White Paper: Advancements in field measurement of Ethernet performance. http://www.flukenetworks.com/support/whitepapers, 2008. Accessed 04-May-2011.

[FO00]. Paul A. Farrell, and Hong Ong. Communication Performance over Gigabit Ethernet Network, In *Proceedings of the 19th IEEE International: Performance, Computing, and Communications Conference (IPCCC'00)*, pp. 181-189, Arizona, February 20-22, 2000.

[FS07] Frank Foerster and Bill Seitz. Ethernet Powerlink: A Deterministic Alternative for Distributed Control. In *the article of RTC Magazine*, April 2007.

[GB02] Paul Gray, and Anthony Betz. Performance Evaluation of Copper-based Gigabit Ethernet Interfaces, In *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks (LCN'02)*, pp. 679-690, Tampa, Florida, November 2002.

[HF04] Justin (Gus) Hurwitz, Wu-chun Feng. End-to-End Performance of 10-Gigabit Ethernet on Commodity Systems, *IEEE Micro*, vol. 24, no. 1, pp. 10-22, Jan./Feb. 2004.

[Hil08] Hilscher Gesellschaft für Systemautomation mbH. User Manual: netSWITCH for SERCOS III. http://www.hilscher.com/files_manuals/ netSWITCH_S3_usermanual_en.pdf, 2008. Accessed 10-January-2011.

[HRB07] Joachim Hillebrand, Mehrnoush Rahmani, Richard Bogenberger, and Eckehard G. Steinbach. Coexistence of Time-Triggered and Event-Triggered Traffic in Switched Full-Duplex Ethernet Networks, *IEEE Second International Symposium on Industrial Embedded Systems - SIES'2007*, Lisbon Portugal, July 4-6, 2007.

[HTG10] HARTING Technology Group. HARTING Ha-VIS Fast Track Switch. http://www.harting.com, 2010. Accessed 30-December-2010.

[IEC00] IEC - International Electrotechnical Commission. IEC 61158 (2000) - Digital data communications for measurement and control - Fieldbus for use in industrial control systems – parts 2 to 6.

[IEC03] IEC – International Electrotechnical Commission. IEC 61784-1 (2003) – Digital data communications for measurement and control: Part 1. Profiles sets for continuous and discrete manufacturing relative to fieldbus use in industrial control systems.

[IEC07] IEC – International Electrotechnical Commission. IEC 61784-2 (2007) Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3.

[IEE] IEEE EtherType Field Registration Authority. http://standards.ieee.org/ develop/regauth/ethertype/eth.txt. Accessed 27-April-2011.

[IEE02] IEEE Standard 1588, 2002 Edition. Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2002.

[IEE04] IEEE Standard 802.1D, 2004 Edition. IEEE Standard for Local and Metropolitan Area Networks – Media Access Control (MAC) Bridges, 2004.

[IEE05a] IEEE Standard 802.3, 2005 Edition. Carrier Sense Multiple Access with Collision Detect on (CSMA/CD) Access Method and Physical Layer Specification, 2008.

[IEE05b] IEEE Standard 802.1Q, 2005 Edition. IEEE Standard for Local and Metropolitan Area Networks – Virtual Bridged Local Area Networks, 2005.

[IEE98] IEEE. Media Access Control (MAC) Bridges. IEEE, New York, 1998. ANSI/IEEE Std 802.1D-1998.

[JE04] Jürgen Jasperneite and Ehab Elsayed. Investigations on a Distributed Time-triggered Ethernet Real-time Protocol used by PROFINET. In *Proceedings of the 3rd International Workshop on Real-time Networks*, pages 69-72, Catania, Italy, July 2, 2004.

[JH08] Endra Joelianto and Hosana. Performance of an Industrial Data Communication Protocol on Ethernet Network. In *Proceedings of the 5th IEEE/IFIP International Conference on Wireless and Optical Communication Networks (WOCN 2008)*, Surabaya, Indonesia, May 2008.

[JISW09] J. Jasperneite, J. Imtiaz, M. Schumacher, K. Weber. A Proposal for a Generic Real-Time Ethernet System. *IEEE Transactions on Industrial Informatics*, vol. 5, no. 2, May 2009.

[JSim] J-Sim - Java library for discrete-time process oriented simulation, http://www.jsim.zcu.cz, Departement of Computer Science and Engineering, University of West Bohemia, Accessed 11-May-2011.

[JSW04] Jürgen Jasperneite, Khaled Shehab, Karl Weber. Enhancements to the Time Synchronization Standard IEEE-1588 for a System of Cascaded Bridges, *5th IEEE International Workshop on Factory Communication Systems (WFCS'2004)*, 2004, pp. 239-244.

[JSW07] Jürgen Jasperneite, Markus Schumacher, and Karl Weber. Limits of Increasing the Performance of Industrial Ethernet Protocols. In *Proceedings of the 12th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 17-24, Greece, Sep. 2007.

[Kac01] Jaroslav Kacer. J-Sim – A Java-based Tool for Discrete Simulations, *Master's Thesis*, University of West Bohemai, Faculty of Applied Sciences, Department of Computer Science and Engineering, Czech Republic, May 2001.

[Kac02] Jaroslav Kacer. Discrete Event Simulations with J-Sim, In *Proceedings of the Inaugural Conference on the Principles and Practice of Programming in Java (PPPJ-2002)*, Ireland, June 2002.

[KAGS05] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The Time-Triggered Ethernet (TTE) Design. *8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, Seattle, Washington, May 2005.

[KBC09] Imad Khazali, Marc-Andre Boulais, and Phil Cole. AFDX Software Network Stack Implementation – Practical Lessons Learned, In Proceedings of IEEE/AIAA 28th Digital Avionics Systems Conference 2009 (DASC ‘09), Orlando, USA, October 23-29, 2009.

[KDDS08] Harald Kleines, Sebastian Detert, Matthias Drochner, and Frank Suxdorf. Performance Aspects of PROFINET IO. In *Proceedings of IEEE Transactions on Nuclear Science*, volume: 55, No. 1, Feb. 7, 2008.

[KG93] Hermann Kopetz, Gunter Grunsteidl. TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems, in *Proceedings of the 23rd IEEE International Symposium on Fault-Tolerant Computing (FTCS-23)*, Toulouse, France, 1993.

[Kop97] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, 1997. ISBN 0-7923-9894-7.

[KSS07] Nikolay Kakanakov, Mitko Shopov, Grisha Spasov, Hristo Hristev. Performance Evaluation of Switched Ethernet as Communication Media in Controller Networks, In *Proceedings of the 2007 International Conference on Computer Systems and Technologies (CompSysTech’07)*, 2007.

[LDK10] Myungjin Lee, Nick G. Duffield, Ramana Rao Kompella. Not All Microseconds are Equal: Fine-Grained Per-Flow Measurements with Reference Latency Interpolation, *the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM’10)*, August 30 – September 3, 2010, New Delhi, India.

[LH04] Jork Loeser, and Hermann Haertig. Low-latency Hard Real-Time Communication over Switched Ethernet, In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS’04)*, pp. 13 - 22, July 2004.

[LL02] Kyung Chang Lee, and Suk Lee. Performance evaluation of switched Ethernet for real-time industrial communications. Computer Standards & Interfaces, vol. 24, no. 5, pp. 411-423, November 2002.

[LL06a] Kyung Chang Lee, and Suk Lee. Worst Case Communication Delay of Real-Time Industrial Switched Ethernet with Multiple Levels. IEEE Transactions on Industrial Electronics, vol. 53, no. 5, pp. 1669 - 1676, October 2006.

[LL06b] Qing Liu and Yingmei Li. Modbus/TCP based Network Control System for Water Process in the Firepower Plant. In *Proceedings of the 6th World Congress on Intelligent Control and Automation (WCICA)*, Volume: 1, pages 432-435, June 21-23, 2006, Dalian, China.

[LM07] Edward A. Lee, Slobodan Matic. Determinism in Event-Triggered Distributed Systems with Time Synchronization, *2007 International IEEE Symposium on Precision Clock Synchronization (ISPCS) for Measurement, Control, and Communication*, Vienna, Austria, October 1-3, 2007.

[LSF09] Xiaoting Li, Jean-Luc Scharbarg, Christian Fraboul. Improving end-to-end delay upper bounds on an AFDX network by integrating offsets in worst-case analysis, 2010 IEEE Conference on Emerging Technology and Factory Automation (ETFA), Sep. 13-16, 2010, Bilbao, Spain.

[MEM08] Bernd Mueller-Rathgeber, Michael Eichhorn, Hans-Ulrich Michel. A unified Car-IT Communication-Architecture: Design Guidelines and prototypical Implementation, Proceedings of the IEEE Intelligent Vehicles Sysposium (IV 2008), Eindhoven University of Technology, Eindhoven, Netherland, June 4-6, 2008.

[NHB05] Thomas Noltet, Hans Hanssont, and Lucia Lo Bellot. Automative communications - Past, Current and Future, *10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, Catania, Italy, Sept. 19-22, 2005.

[Obe05] Roman Obermaisser. *Event-Triggered and Time-Triggered Control Paradigms*. Springer Science + Business Media Publisher, 2005, ISBN 0-387-23044-0.

[OB09] Onur Özgün, Yaman Barlas. Discrete vs. Continuous Simulation: When Does It Matter?, In *Proceedings of the 27th International Conference of the System Dynamics Society*, July 26-30, 2009, Albuquerque, NM, USA.

[Odv06] ODVA – Open DeviceNet Vendors Association. *EtherNet/IP^{TM} – CIP on Ethernet Technology*. http://www.odva.org/Portals/0/Library/ Publications_Numbered/PUB00138R2_CIP_Adv_Tech_Series_EtherNetIP.pdf, 2006. Accessed 20-December-2010.

[OSI96] OSI - Open Systems Interconnection. Basics Reference Model: The Basic Model. ISO/IEC Standard. 7498-1, 1994.

[PAS06] P. Ferrari, A. Flammini, and S. Vitturi. Performance Analysis of PROFINET networks, *Journal Computer Standards and Interfaces*, Vol. 28, Issue 4, April, 2006.

[Pau01] Brooks Paul. *EtherNet/IP: Industrial Protocol White Paper*, IEEE EFTA 2001, Oct. 2001.

[PI09] PROFINET International. *PROFINET Technology and Application – System Description*, http://www.profibus.com/nc/downloads, English version, 2009. Accessed 3-January-2011.

[PJ05] Gunnar Prytz, and Svein Johannessen. Real-time Performance Measurements using UDP on Window and Linux, *In Emerging Technologies and Factory Automation – ETFA*, Catania, Italy, 2005.

[Pry08] Gunnar Prytz. A Performance Analysis of EtherCAT and PROFINET IRT. In Proceedings pf the 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), September 2007.

[Puz02] Rita Puzmanova. Routing and Switching. Addison-Wesley, 2002. ISBN 0-201-39861-3.

[Rug08] RuggedCom Incorporated. Latency on a Switched Ethernet Network. http://www.ruggedcom.com/pdfs/application_notes/latency_on_a_switched_ethernet_network.pdf, 2008. Accessed 04-May-2011.

[SBH09] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, S.Varadarajan. TTEthernet Dataflow Concept. *2009 Eighth IEEE International Symposium on Network Computing and Applications*, pp. 319-322, July 9-11, 2009.

[SB08] Wilfried Steiner, Günther Bauer. ETHERNET FOR SPACE APPLICATION: TTETHERNET, *International SpaceWire Conference 2008*, Nara, Japan, November 4-6, 2008.

[Sch07] Detlev Schaadt. AFDX/ARINC 664: Concept, Design, Implementation and Beyond, *SYSGO AG White Paper*, 2007.

[Sei98, pp.120] Rich Seifert. Gigabit Ethernet: Technology and Applications for High-Speed LANs, Addison-Wesley, 1998.

[SER07] SERCOS International. SERCOS III Universal Real-Time Communication with Ethernet. http://www.sercos.com/literature/pdf/sercos3_en.pdf, 2007. Accessed 10-January-2011.

[SGAK06] Klaus Steinhammer, Petr Grillinger, Astrit Ademaj, and Hermman Kopetz. A Time-Triggered Ethernet (TTE) Switch. In Proceedings of *2006 Design, Automation and Test in Europe (DATE '06)*, Munich, Germany, 6-10 March 2006.

[SHF06] SHF Communication Technologies AG. EPLGW – Ethernet to Ethernet Powerlink Gateway (Revision 1.3). http://www.shf.de/fileadmin/download/Automation/h_epg_0609_005en.pdf, September 2006. Accessed 19-January-2011.

[SJR05] Jason Schreiber, Mehradad Khodai Joopari and M. A. Rashid. Performance of video and video conferencing over ATM and Gigabit Ethernet backbone networks, *Res. Lett. Inf. Math. Sci.*, 2005, Vol. 7, pp. 19-27.

[SMJ08] Wilfried Steiner, Reinhard Maier, David Jameux, Astrit Ademaj. Time-Triggered Services For SpaceWire. SpaceWire Conference, 2008, Nara, Japan.

[Spu00] C.E. Spurgeon, Ethernet: The Definitive Guide. Sebastopol, CA: O'Reilly, Feb. 2000.

[Ste06] Klaus Steinhammer. PhD. Dissertation. Design of an FPGA-Based Time-Triggered Ethernet System. Institute of Computer Engineering, Vienna University of Technology, December 5, 2006.

[SV08] Teresa Schuster, Dinesh Verma. Networking Concepts Comparison for Avionics Architecture, IEEE/AIAA 27th Digital Avionics Systems Conference (DASC 2008), Minnesota, USA, Oct. 26-30, 2008

[TTE08] TTE Specification. http://www.tttech.com, August 2008.

[TTTech] TTTech Computertechnik AG, Wien. Available: http://www.tttech.com

[Ver94] P. Verissimo. Ordering and Timeliness Requirements of Dependable Real-Time Programs. *Real-Time Systems*, Vol. 7(3), pp. 105-128, 1994.

[VVT06] Raimundo Viegas Jr, Ricardo A. M. Valentim, Daniel Garcia Texeira, and Luiz Affonso Guedes. Performance Measurements of Protocols to Ethernet Real-Time Applications, *SICE-ICASE International Joint Conference 2006*, Busan, Korea, October 18-21, 2006.

[WK06] L. Winkel and Karlsruhe, Real-Time Ethernet in IEC 61784 and IEC 61158 Series. *IEEE International Conference on Industrial Informatics*, Singapore, 2006, pp. 246-250.

Curriculum Vitae

Personal details

Name:	Ekarin Suethanuwong
Gender:	Male
Nationality:	Thai
Birth:	20-April-1977
Place of Birth:	Bangkok Thailand

Education

2007 – Present	Vienna University of Technology (TU Wien, Austria) Institute of Computer Engineering, Real-Time Systems Group PhD: “Performance Evaluation and Optimization of Standard-Ethernet Traffic in TTEthernet Systems”
2002 – 2005	Prince of Songkla University (Thailand) Master’s Degree in Computer Engineering
1995 – 1999	Prince of Songkla University (Thailand) Bachelor’s Degree with second-class honor in Electrical Engineering

Professional Experiences

2005 – 2007	Lecturer in Management Information Technology Faculty Commerce and Management Prince of Songkla University (Trang Campus) in Trang (Thailand)
2001 – 2002	Hardware Engineer Mastertech International Co., Ltd. in Bangkok (Thailand)
2000 – 2001	Test Engineer NS Electronics Bangkok (1993) Co., Ltd. in Bangkok (Thailand)
1999 – 2000	Electrical Engineer Charoen Pokphan Foods Public Co., Ltd. in Samutsakorn (Thailand)