The Rationale for Time-Triggered Ethernet

Invited Keynote Paper
Hermann Kopetz
TU Wien, Austria
hk@vmars.tuwien.ac.at

Abstract

Time-Triggered (TT) Ethernet is a new real-time communication protocol that is fully compatible with Ethernet and provides in addition to the standard Ethernet service a deterministic real-time communication service for distributed real-time systems. This paper elaborates on basic concepts in real-time communication, elicits real-time communication requirements and discusses some innate conflicts that must be considered in any real-time protocol design. In the second part, the rationale and the principles of operation of TT-Ethernet are presented and common properties of all members of the TT-Ethernet protocol family are discussed.

1. Introduction

A real-time communication service is a fundamental service in any distributed real-time computing system. For this reason a plethora of different application-domain-specific real-time communication protocols have been proposed over the past decades, such as CAN[1], Profibus[2], many variants of real-time Ethernet [3], AFDX [4], TTP [5], and FlexRay[6], to name a few. However, as of today, none of these protocols has achieved such a universal acceptance in the real-time community as standard Ethernet[7] has received in the non-real-time world.

In our research group at the Technical University of Vienna we have worked in the field of real-time protocol design for more than twenty years. Our first real-time protocol, the time-triggered protocol TTP/C [5] is now in industrial use in the aerospace sector. For example, a number of onboard control systems in the Airbus A380 and in the Boeing 787 aircraft are communicating by TTP. In the meantime the automotive industry has developed its own version of a time-triggered protocol, FlexRay that is based on the experience with TTP. FlexRay will be used in future automotive systems. We firmly believe that time-triggered protocols will find their use in many other application domains.

We feel that the technological and economic developments of the last decade will enforce a consolidation in the real-time protocol arena. At a time, when many different application domains are becoming integrated and when the cost of a new silicon implementation of a hardware protocol controller on a system-on-chip (SoC) is becoming very expensive, it is not sustainable to support twenty or more different incompatible real-time protocols. According to our assessment, a successful real-time protocol must have the following properties:

It must be based on sound theoretical foundations with respect to time, determinism and composability.

It must support all types of real-time applications, from mass-market multi-media applications, to professional control systems in different application domains, to safety-critical applications that require fault-tolerance and certification.

It must be economically competitive—protocol controllers that are integrated on a System on Chip should cost less than one Euro.

It must support non-real-time traffic preferably along the Ethernet standard [7] that is widely used in the non-real-time world.

Let us comment on this last point. We feel that Ethernet applications have passed the *tipping point* in the non-real-time domain. Even the new optical networks, such as GPON [8], use the data format of the Ethernet standard.

Based on our experience in the design of real-time protocols and the industrial feedback we received from the applications of TTP we decided to develop a novel real-time protocol that meets the above stated requirements. In the protocol design, special emphasis has been place on the issue of *cognitive simplicity* [9]. This new protocol is called *TT (time-triggered) Ethernet*.

1052-8725/08 $25.00 © 2008 IEEE
DOI 10.1109/RTSS.2008.33

The rest of the paper is structured as follows. In the next Section we elaborate on and clarify the basic concepts that are needed to discuss and analyze distributed real-time communication systems. Section three discusses the requirements of real-time protocols and investigates some innate conflicts that must be considered in any real-time protocol design. Section four presents the rationale and the principles of operation of time-triggered TT-Ethernet. Section five presents common properties that apply to all members of the TT-Ethernet protocol family. The paper terminates with a conclusion in Section six.

2. Basic Concepts

In this Section we elaborate on those concepts that must be clearly defined before any analysis of distributed real-time communication systems can be performed.

2.1. Time

When we use the word *time* in this paper we mean physical time as defined in the international standard of time TIA[10]. A cut of the timeline is called an *instant*. The interval between two instants is called a *duration*. An occurrence at an instant is called an *event*. In this paper the words *real-time* and *time* are used synonymously—they denote the same concept. In order to be able to establish a system wide consistent notion of simultaneity of *distributed events* we consider a *global sparse time base* as a key concept in spatially distributed computer systems.

In the *sparse-time model* the continuum of real-time is partitioned into a sequence of alternating intervals of *activity* of duration ε, and *silence* of duration Δ. The duration of these intervals depends on the *precision* of the clock synchronization[11]. Within the system, the occurrence of events of significance (e.g., the *sending of a message)* is restricted to the *activity intervals.* We call the events that occur within an activity interval *sparse events*. We number the activity intervals by positive integers and call the integer number assigned to an activity interval, the *global timestamp* (or *timestamp* for short) of events happening in that interval. We consider all sparse events that happen within the same *activity interval* ε as having occurred *simultaneously.* A *system-wide consistent temporal order* of all *sparse events* occurring in a distributed system can now be established on the basis of their global timestamps. We assume that all events that are in the *sphere of control* of the system (e.g., the sending of messages) happen within the activity intervals ε of the sparse time base and are therefore *sparse events*. Events that are outside the sphere of control of the system and occur on a dense time base must be transformed to *sparse events* by the execution of an agreement protocol[11]. The granularity of the sparse time-base must be chosen in agreement with the achievable precision of the clock-synchronization at the selected abstraction level.

2.2. Cycle

A *cycle* is a period of time between the repetitions of regular events. A cycle is specified by the duration of its *period* (or its frequency) and the position of its start, the *cycle phase,* relative to some given global time reference. The cyclic representation of time (in contrasts to the linear representation of time) is ideal to represent progress in periodic processes.

2.3. Real-Time (RT) Cluster

Any distributed real-time computer system consists of a set of *cooperating computing components* that exchange real-time messages among each other and communicate with the environment to achieve the stated purpose. We call the set of cooperating computing components that are connected by a real-time communication system a *RT-cluster*. The *controlled object*, the *human operator* and/or *other computing systems* form the environment of a RT-cluster.

2.4. Component

A *component* is a hardware/software unit that accepts input messages, provides a useful service, maintains internal state, and produces after some *elapsed time* output messages containing the results. A component is thus an identifiable functional unit of data transformation and comprehension and forms an abstract high-level concept in the mental model of system behavior.

2.5. Message

A *message* is an atomic data structure that is formed for the purpose of *transmitting data* and *control signals* from a sending component at a given instant to one or more receiving components that receive the message at a later instant. The message concept does not make any assumption about (abstracts form) the physics of the specific transport mechanism (e.g., wire bound or wireless transmission) or about the meaning of the bit-vector contained in the data field of the message. However, the time it takes to transport a message from the sending component to the receiving component is part of the control aspect of the message concept. A message that is intended to arrive at its receiver(s) within a given time-interval is called a *real-time message*; otherwise it is a *non-real-time message*. A real-time message is *correct* if it contains the *intended* data in its data field and is sent and received at the *intended* time.

2.6. Real-time data

The bit-vector that is contained in the data-field of a message represents *real-time data*, if the meaning (semantics) of this data depends on the instant of observation of the entity of relevance. For example, if we measure the position of a piston in an automotive engine this measurement—we call it an *observation*—is only meaningful if we consider an *observation* as an *atomic triple* formed by the *instant of observation*, the *observed value* and the *name of the entity* that is observed[11]. Taking it to the extreme, all data about the world is real-time data. However, if the change of the data (its dynamics) is considerable slower than the time-constants involved in processing the data, we often abstract from the temporal dimension and consider the data to be non-time dependant.

2.7. Behavior and Service

We call the temporal sequence of messages that are produced and consumed by a component the *behavior* of the component. The *intended* behavior of a component is called its *service*. A service consists of the temporal sequence of correct input and output messages. The syntax and semantics of the component service should be specified in a component-interface model without reference to the detailed component internals, i.e., the concrete component implementation. In real-time systems the interface model must include the temporal parameters of the intended component behavior. Preferably, the interface model should specify the data-transformation algorithms of the component in an executable form, such that the algorithms can be automatically translated into the selected implementation technology. From the point of view of time, a service is called *best-effort*, if the interval taken between the service request and the service response cannot be predicted. A service is called *predictable*, if the interval between service request and service response is known *a priori*.

2.8. Open vs. Closed Communication

We call a communication scenario *open*, if the sending components can act independently, can join and leave the scenario dynamically and can try to send messages at any instant. An example for an open scenario is standard Ethernet[7]. We call a communication scenario *closed* if the sending components cooperate in order to avoid temporal conflicts in the communication system. An example for a closed scenario is any time-triggered protocol, such as TTP[5] or FlexRay[6].

2.9. State

The concept of *state* is introduced in a model of a system in order to separate past behavior from future behavior. We follow the definition of Mesarovic[12], p.45 :

The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other word, the state enables a "decoupling" of the past from the present and future. The state embodies all past history of a system. Knowing the state "supplants" knowledge of the past. . . . Apparently, for this role to be meaningful, the notion of past and future must be relevant for the system considered.

The sparse time model introduced above makes it possible to establish the consistent system-wide separation of the past from the future that is necessary to define a consistent system state between selected (periodic) activity intervals in a distributed system. Without a proper model of time, the notion of state of a distributed system is an *ill-defined* concept.

2.10. Determinism

Our first definition of *determinism* is derived from *[13]*: *A model behaves deterministically if and only if, given a full set of initial conditions (the initial state) at time t_o, and a sequence of future timed inputs, the outputs at any future instant t are entailed.*

This definition of determinism is intuitive, but it neglects the fact that in a real (physical) distributed system clocks cannot be precisely synchronized and therefore a system-wide consistent representation of time (and consequently state) cannot be established. We therefore need a revised, more realistic definition of determinism in a distributed real-time computer system: *A model of a distributed computer system (hardware, software, communication) is said to behave deterministically if and only if, given a sparse time-*

base with an infinite sequence of intervals t_j, the state of the system $\Sigma(t_0)$ at t_0 (now), and a set of future Input Messages $IM_1(t_{i1})$, $IM_2(t_{i2})$, ... , $IM_n(t_{in})$, then the set of future Output Messages $OM_1(t_{o1})$, $OM_2(t_{o2})$, ... , $OM_n(t_{on})$ and the state of System $\Sigma(t_x)$ at all future intervals t_x is entailed.

3. Real-Time Communication

3.1. Requirements

A distributed real-time system normally requires two different kinds of communication services: (i) *a best-effort communication service* that is used to transport *non-real-time messages* among the components of the cluster and to computers outside the cluster and (ii) a *predictable communication service* that transports *real-time messages* from the sending to the receiving components within a cluster.

Best effort communication service: A best-effort communication tries to deliver the messages from the sender to the receiver in the shortest interval, given the current constraints in the communication system that are changing dynamically. For example, in switched Ethernet, a message is forwarded immediately to the receiver if the channel to the receiver is free. In case this channel is busy, the message is stored in the switch until the channel becomes free again. Best-effort communication services are provided for an *open communication scenario*.

Predictable communication service: A predictable communication service delivers a message from a sender to its receiver within an *a priori* known interval. A predictable communication service is needed, if *real-time data* has to be exchanged among the nodes of a cluster. The preferred communication mechanism for the exchange of real-time data is the *deterministic multicast unidirectional message.*

Determinism is needed for the following reasons:

(i) *Timeliness*: Many embedded systems require timely responses. The notion of *determinism* subsumes predictable timing (see Section 2.1).

(ii) *Complexity reduction:* It is much easier to reason about the behavior of a communication system, if the message transport is deterministic, i.e. it is exactly known at what instant a message will arrive, than if the behavior of the communication system is *best-effort*[9].

(iii) *Testing*: The testability of a system is improved, if the system will produce the same outputs given it has been offered identical inputs[14].

(iv) *Active Redundancy*: The implementation of active redundancy requires a deterministic behavior of the replicated components.

Multicast communication is needed in order to enable the observation of the behavior of a component by an independent external observer—a monitor—without introducing the *probe effect* [14].

Uni-directionality supports the strict separation of communication from computation and thus supports the *simplification strategy of partitioning*. If the basic communication service were *bi-directional* (e.g., *request-response*) then the separation of the communication service from the processing at the receiver were not supported. We feel that one of the reasons why the basic architecture of the *Internet* is scalable and has survived a growth in the number of connected computers by more than six orders of magnitude is the unidirectional communication service coupled with the *fate-sharing* principle[15].

3.2. Innate Conflicts

If we analyze the above listed requirements for a real-time communication service, we find some *innate conflicts* that must be addressed and resolved in the design of any real-time communication protocol.

Open vs. Closed Communication Scenarios: *It is impossible to provide temporal guarantees in an open communication scenario*. If every sending component in an open communication scenario is autonomous and is allowed to start sending a message at any instant, then it can happen that all sending components send a message to the same receiver at the same instant (the *critical instant*), thus overloading the channel to the receiver. In fielded communication systems, we find three strategies to handle such a scenario:

(i) The communication system delivers one message at a time and stores the other messages in the communication system until the channel to the receiver becomes available again. This strategy is followed in standard switched Ethernet[7].

(ii) The communication system exerts *back-pressure* on the sender and forces the sender to wait until the channel to the receiver becomes available again. This strategy is followed in bus-based Ethernet or in the CAN[1] system.

(iii) The communication system accepts some messages and discards the rest. This strategy is followed in switched Ethernet in case when there is a buffer overflow in the switch.

All three strategies are problematic from the point of real-time performance. Only in a closed communication scenario, where the senders cooperate to avoid temporal conflicts in the communication system, can we provide a predictable communication service. If a global notion of time of known precision is available in all sending components, this global time can be used to coordinate the senders.

Consistent Ordering of Events: *It is impossible to establish on the basis of their timestamps a consistent view of the temporal order of events in a distributed system, if the events can be generated at any instant of a dense time-line.* Consider the case, where the occurrences of two independent events have to be communicated among the components in the cluster by two different observers using replicated channels. If these two events occur simultaneously (at the same instant), their assigned time-stamps can differ by one tick, even if a reasonable granularity has been chosen for the global time[11]. The two sending nodes might thus establish a different temporal order of the messages related to the two events. These two orders are then *inconsistent*. One way out of this dilemma is to establish a sparse time base and to limit the sending of messages to the active intervals of the sparse time base as discussed in the previous Section.

Single synchronization domain: *It is impossible to support more than a single synchronization domain for the temporal coordination of components within a RT-cluster.* Consider the example where a highly dynamic scenario is observed by two or more smart cameras every 50 msec to capture the scene and perform scene analysis. In order to eliminate the effects of the changing environment, the two cameras should take their picture at about the same instant. Let us assume that both cameras are self-timed. Even if the clocks in the cameras are of high quality, it will happen over time that the phases of the two cameras will start to drift apart and the requirement that the pictures must be taken at about the same instant is violated. This problem can be solved if a single synchronization domain is established: one camera can be self-timed and the actions of the other camera have to be synchronized to the time established by the self-timed camera.

Of course, it is possible to introduce multiple RT clusters in a large system, where each RT cluster forms its own autonomous synchronization domain. As a consequence, the control actions among these autonomous clusters are not temporally coordinated in such a system.

Determinism: *It is impossible to build a deterministic distributed real-time system without the establishment of some sort of a sparse global time base for the consistent time-stamping of distributed events.* Without a sparse global time base, simultaneity cannot be resolved consistently in a distributed system, possibly resulting in an inconsistent temporal order of the messages that report about these simultaneous events. Inconsistent ordering results in the loss of *replica determinism*[16]. The assignment of events to a sparse global time-base can be established at the system level by the introduction of *sparse events* or at the application level by the execution of agreement protocols.

3.3. Message Categories

The above-discussed conflicts can be resolved if different message categories with differing properties are introduced in the communication system for the transport of real-time data and non-real-time data.

Event-triggered Message: The event-triggered control schema is the usual control schema associated with the message concept. It can be used to send non-real-time data. An *event-triggered message* is sent whenever a significant event occurs at the sender. Event-triggered message must conform with the *exactly-once semantics*, i.e., every message produced by a sender must be eventually consumed *exactly once* by its receiver(s). In case the communication channel is not free at the instant of event-occurrence, the event-triggered message must be stored in a queue at the sender's site before it can be transmitted by the network. Similarly, an event-triggered message that is delivered by the network to the destined receiver must be stored in a message queue before the receiver in case the receiver is not ready to accept the message at the instant of message arrival. Thus, there are (at least) two queues associated with every event-triggered message, one at the sender and one at the receiver. The sizing of these queues depends on the uniformity and rate of message production at the sender, the uniformity and rate of message consumption at the receiver and the available capacity of the communication channel. For example, if we have a large-bandwidth channel, the queue at the sender will be small and the queue at the receiver will be large. If we have a small bandwidth channel, it may be the other way around.

We distinguish between two types of event-triggered messages, depending on the bandwidth allocation to the communication channel. If a fixed (static) bandwidth is assigned to every event-triggered channel, we call the communication channel *predictable*. In a system with predictable event-triggered communication channels, the sizing of the two queues can be performed in the local context of sender and receiver. An example of a communication system with a predictable channel is a channel with constant bandwidth (e.g., a TDMA channel) that is used for the transmission of a single event-triggered message type.

If the bandwidth assigned to a sender is dynamic, i.e. depending on the activity of other senders that use the same communication channel, then we call the communication channel *best effort.* In a system with a best-effort channel, the sizing of the two queues can be performed only in the global context of all users of the channel. In such a system composability with respect to temporal properties is in danger. Examples for a communication system with best-effort channels are standard Ethernet or the CAN [1] protocol, that is widely used in the automotive environment.

Since it is difficult to assure that a best-effort unidirectional channel will deliver a message within a given time-interval, a higher-level protocol that binds two event-triggered channels together is commonly formed to be able to inform the sender of the successful receipt of the message and realize a time-constrained error-detection service in case a message does not arrive within the given protocol-specific time interval.

It is difficult to give precise temporal guarantees for event-triggered messages, since the delay of the event-triggered messages in the sender queue and the receiver queue is difficult to quantify, even if the communication system is predictable.

Time-triggered Message: A cycle is assigned to every time-triggered message. The *time-triggered message* is sent whenever the *cycle start* of the cycle that has been assigned to this message occurs. The cycles assigned to the time-triggered messages must be planned *a priori* by a *message scheduler*, in order to avoid any message conflicts among time-triggered messages. At the instant of *cycle start* of a time-triggered message, the contents of the message buffer at the sender's site are fetched by the communication system and transmitted to the receivers (non-consuming read) within an *a priori* know interval. At the instant of message delivery by the communication system, the content of the message buffer at the receiver's site is overwritten by the arriving time-triggered message. There are no queues associated with time-triggered messages. Since the time-triggered communication system is free of conflicts and deterministic by design it is possible to associate temporal guarantees with time-triggered messages. Time-triggered messages are well-suited to transmit data items with state semantics in periodic control systems.

From a conceptual point of view, communication by time-triggered messages is easy to comprehend because a time-triggered message provides a powerful abstraction of the component's environment—a *temporal firewall* to the component environment[17]. At the receiver's side the message buffer of a (periodic) time-triggered message always contains an *image* of the most-up-to-date value of a remote state variable. This value can be accessed locally just like the value of any other local variable. The message buffer of a time-triggered message provides the only interface to the external world and eliminates control-error propagation from the external environment into the component by design. The cycle of the time-triggered message determines the worst-case temporal validity—the maximum age—of the accessed value.

In a time-triggered system, the detection of a lost or corrupted message can be performed by the receiver on the basis of the *a priori* knowledge about the expected arrival time of the periodic time-triggered messages. Examples for time-triggered protocols are TTP[5] FlexRay [6] or time-triggered (TT) Ethernet [18].

Data Stream: A *data stream* is a regular sequence of messages that is produced by a sender and consumed by a receiver. Data streams are important in multimedia systems. It is possible to perform *on-the-fly* processing of data streams in the temporal interval between messages. Data streams have the temporal properties (*real-time transmission*) of time-triggered messages and value properties (*exactly-once-semantics*) of event-triggered messages. In a time-triggered architecture it is possible to synchronize data-streams from different sources which makes this architecture well-suited for multi-media applications (e.g., lip synchronization by synchronizing an audio stream with a video stream).

Since the production rate and consumption rate of multimedia messages is often content dependent, it is common to associate *watermarks* with a data stream to monitor the length of the sender and receiver queues. A higher-level protocol sends these watermarks from the receiver to the sender in order to inform the partners about the fill-level of the queues and alert the producer of messages about the need to adapt the production rate of messages.

The precise synchronization of the producer and consumer of multimedia messages reduces the need for intermediate buffers and saves storage and energy, which is of special importance in portable devices.

4. Principles of Operation

Based on the insight gained from the previous analysis, we give the rationale for and explain the principles of operation of TT-Ethernet. TT-Ethernet is designed to become a generic communication system that can be used both in real-time applications and in non-real-time applications.

4.1. Core Idea

In addition to traffic categories already used in standard Ethernet systems, such as best-effort and rate-constrained traffic, (from now on, we call these standard Ethernet Messages *ET messages,* because they are event-triggered) TT-Ethernet introduces the new category of time-triggered Ethernet messages (*TT-Messages*). From the point of view of the frame structure and addressing conventions these two message categories are identical. The only difference is in the control, i.e., the timing of the message transport by the TT-Ethernet switch. An ET message is sent across the communication system whenever the sender decides to send an ET message. The TT-Ethernet switch forwards the ET message according to the best-effort strategy of standard Ethernet to its receivers. A TT-Message that arrives at the switch is delayed and stored in the switch until the *a priori* determined *cycle start* that is associated with this TT-Message has occurred. As soon as the *cycle start* instant happens, the message is forwarded to its receivers within the a priori known constant delay. In our 100 Mb prototype implementation of a TT-Ethernet switch, this constant delay is less than five microseconds.

Since the ET Message (the standard Ethernet messages) originates from an open scenario of uncoordinated senders temporal conflicts in the Ethernet switches cannot be avoided. This can lead to a possible unknown delay of the ET message in the switch. It can also happen that during the transmission of an ET message a scheduled TT-Message arrives at the switch. In this case the switch will delay or *preempt* the transmission of the current ET message and will transport the TT-Message within its constant *a priori* known delay. After the transmission of the TT-Message has been finished, a preempted ET message (which is stored in the switch until its transmission is successful) will be retransmitted by the switch. This strategy results in a possible increase of the delay of an ET message and in a constant guaranteed delay for TT-Messages. Since the Ethernet standard does not mention any temporal guarantees—this is *impossible* in an open communication scenario—this strategy of extending the delay of ET message transmissions is fully compliant with standard Ethernet.

4.2. Identification of a TT-Message

As mentioned before, the data field and the addressing conventions for messages are the same for ET-Ethernet and TT-Ethernet. How then can a TT switch identify an incoming message as a *TT-Message*? There are three means for the switch to perform this identification: (i) by examining the contents of the Ethernet type field or (ii) by examining the contents of the address fields that are contained in every Ethernet message or (iii) by monitoring the time window during which a new message arrives at the switch or by a combination of the above means

The *Ethernet type field* is a standard two-byte field in the header of every Ethernet message. If its value is in the range from *hex 0000* to *hex 05DC*, the Ethernet standard specifies that the content of the type field denotes the length of the message. If the value of the type field is larger than *hex 0200*, the field denotes a special type of Ethernet message. The Ethernet standard authority from the IEEE assigns dedicated bit patterns for this field in order to uniquely identify special Ethernet message. The IEEE standard authority has assigned the bit pattern *hex 88d7* to identify a TT-Message.

There are also other ways to indentify a TT-Message. In order to make TT-Ethernet compatible with higher-level protocols that already use the *Ethernet type field*—e.g., the IEEE 1588 time synchronization protocol [19]—it is possible to identify TT-Messages by examining the contents of the Ethernet address fields or by monitoring the instant of arrival of a message at the TT switch.

4.3. Synchronization

According to Section 2.3 a RT-cluster can only comprise a single synchronization domain. If multiple synchronization domains are needed in a large real-time application, each one of these synchronization domains will form its own TT-Ethernet cluster. The multiple synchronization domains can communicate by the exchange of ET-Ethernet messages.

Depending on the capabilities of a TT-Ethernet cluster, three different synchronization strategies are supported: (i) data-source synchronization, (ii) time synchronization, and (iii) fault-tolerant time-synchronization.

Data-source synchronization: In the data-source synchronization mode, the TT-Ethernet switch assumes that the TT-data sources are all externally synchronized (e.g., at the application level outside the TT-Ethernet protocol) to form a sparse time- base and that the data sources will not send TT-Messages that are in temporal conflict with each other. The switch itself is *not time aware* and does not contain any internal state. Every incoming message that is identified as a TT-message will be immediately transported to its destination by the switch. An ET-message, which is in conflict with an incoming TT-message will be delayed or preempted and sent to its destination after the channel becomes free again. If two incoming TT-messages are in conflict, the first one will be delivered, the later message will be discarded and an error will be reported. In this synchronization mode, timing failures of components are not contained by the TT-Ethernet switch.

Time synchronization: It is possible to synchronize a TT switch and all connected components to an external timing source. This timing source can be either a (single) self-timed data source or an external time reference, such as the IEEE 1588 standard on time synchronization. This time-synchronization makes the TT-switch time aware.

After startup of a time-synchronized TT-Ethernet switch, the switch must receive its *configuration state*. The *configuration state* comprises information as how to identify the TT-messages, tells the switch at which one of its input ports an identified TT-message is expected to arrive and what cycle (period and phase) is associated with this TT-Message. This configuration state can be sent to the switch by a standard (ET) Ethernet message. The message schedule contained in the configuration state is planned by an off-line or on-line scheduler and can be either static or dynamic. In order to avoid the exponential explosion in the combination of cycles and thus a massive increase in the complexity of a dynamic run-time message scheduler, the rule can be followed that all cycles must be in a *harmonic relationship*, i.e., the duration of any cycle must be a power of two of the duration of the shortest allowed cycle. Whenever a new schedule has been calculated in a dynamic TT-Ethernet system, a new configuration message must be sent to the TT switch. In a dynamic system, where the switch state is stored in volatile memory, it is expedient to send the current configuration state to the switch periodically in order that the switch can recover quickly after a transient fault has corrupted its internal state.

A TT switch which employs time synchronization erects a temporal firewall with respect to timing failures of a sending component. Every identified TT-Message will be sent to its destination exactly at the cycle start instant. Since a TT-Message must arrive earlier than the cycle start instant, it will be stored and delayed in the switch until the cycle start instant. On sending, the message will be consumed. If no new message is available at the cycle start instant, no message will be sent. In such a system, an arbitrary timing failure of a any component—except the time source—will be contained in the switch interface and cannot propagate to and disturb the other components of the cluster.

Fault-tolerant time synchronization: In high-end TT-Ethernet systems it is possible to implement a fault-tolerant time synchronization, such that the timing source itself does not contain any single point of failure.

5. TT-Ethernet Family

TT-Ethernet is not a single protocol, but a family of upwards compatible protocols that support at one end dynamic low-cost multimedia applications with limited error-detection capabilities and at the other end fault-tolerant safety-critical applications that contain a fault-tolerant time synchronization, support fault-masking mechanisms, such as triple-modular redundancy, and contain security mechanisms. In many instances, these high-end systems need to be certified.

The following list of properties is common to all members of the TT-Ethernet protocol family.

ET-messages: All TT-Ethernet configurations support the IEEE Standard 802.3 for the transmission of standard (ET) Ethernet messages.

Communication controller: A standard Ethernet communication controller, as it is available in many systems on chip, can be used to send and receive ET and TT-Ethernet messages.

Data field: The data field of ET and TT-Messages are exactly alike. The difference between ET messages and TT-Messages is in the control (timing) by the switch, not in the data field.

Addressing: The addressing conventions of TT-Messages and ET messages are same and conform to the standard Ethernet addressing conventions established in the IEEE Ethernet standard.

Time format: Whenever time is represented in a TT-Ethernet system, the time format conforms to the time-format of the IEEE 1588 time standard[19].

Determinism: The transport of TT-Messages by a TT switch is deterministic.

These properties that are common to all members of TT-Ethernet protocol family, make it possible to support the gradual evolution of applications, starting from a low-end TT-Ethernet configuration to high-end safety-critical design without an architecture break.

6. Conclusion

The development of any real-time protocol must be aware of the innate conflicts that have to be resolved in the design of any communication protocol for distributed real-time systems. Time-triggered Ethernet addresses these innate conflicts by introducing two message categories, standard messages and real-time messages and a sparse global time base that is needed to provide a deterministic communication service.

Acknowledgements

This work was supported in part the EU Project GENESYS under project number FP 7/213322. Constructive comments to an earlier version of this paper by B. Huber, W. Steiner and G. Stoeger are warmly acknowledged.

References

[1] CAN, "Controller Area Network CAN, An In-Vehicle Serial Communication Protocol", in SAE Handbook 1992. 1990, SAE Press. p. 20.341-20.355.
[2] Profibus, "The Profibus Standard", Profibus Nutzerorganisation, e.d., Hersler Strasse 31, D-503689 Wesseling, 1992.
[3] Real-Time Ethernet, "Information about Real-time Ethernet in Industry Automation", 2004, Available from: http://www.real-time-ethernet.de/.
[4] AFDX, "Avionics Full Duplex Switched Ethernet – ARINC Standard 664", Aeronautical Radio Incorporated, Annapolis USA, 2003.
[5] Kopetz, H. and G. Gruensteidl, „TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems", in Proc. 23rd IEEE International Symposium on Fault-Tolerant Computing (FTCS-23), 1993, Toulouse, France: IEEE Press.
[6] Berwanger, J., et al., "FlexRay: The Communication System for advanced automotive control systems", SAE Transactions, 2001. 110(7): p. 303-314.
[7] Ethernet. "IEEE Ethernet Standard 802.3", Available at URL: *http://standards.ieee.org*, 2002.
[8] GPON, "Gigabit passive optical network", 2008: Wikipedia, the online encyclopedia.
[9] Kopetz, H., "The Complexity Challenge in Embedded System Design", in 6th ISORC, 2008, Orlando, Fl: IEEE Press.
[10] TAI, "International Atomic Time", 2008: Wikipedia, the online encyclopedia.
[11] Kopetz, H., "Real-Time Systems, Design Principles for Distributed Embedded Applications"; ISBN: 0-7923-9894-7, Seventh printing 2003, 1997, Boston: Kluwer Academic Publishers.
[12] Mesarovic, M.D. and Y. Takahara, "Abstract Systems Theory", Lecture Note in Control and Information Science, Vol. 116. 1989: Springer Verlag.
[13] Hoefer, C., "Causality and Determinism: Tension, or Outright Conflict", Revista de Filosofia, 2004. 29(2): p. 99-225.
[14] Schütz, W., "The Testability of Distributed Real-Time Systems", Vol. ISBN 0-7923-9386-4. 1993, Boston, MA: Kluwer Academic Publishers. 160.
[15] Clark, D. "The Design Philosophy of the DARPA Internet Protocols", in ACM Sigcomm Computer Communication Review. 1988.
[16] Poledna, S., "Fault-Tolerant Real-Time Systems, The Problem of Replica Determinism", Vol. ISBN 0-7923-9657-X. 1995, Hingham, Mass, USA: Kluwer Academic Publishers.
[17] Kopetz, H. and R. Nossal, "Temporal Firewalls in Large Distributed Real-Time Systems", in Proceedings of IEEE Workshop on Future Trends in Distributed Computing. 1997. Tunis, Tunesia: IEEE Press.
[18] Kopetz, H., et al., "The Design of TT-Ethernet", in ISORC 2005. 2005. Seattle: IEEE Press.
[19] IEEE, "IEEE 1588, Standard for a Precision Clock Synchronization Protocol for Network Measurement and Control Systems", 2002.