Advancements in Dependable Time-Triggered Communication

Wilfried Steiner

TTTech Computertechnik AG
Vienna, Austria
wilfried.steiner@tttech.com

Abstract. When developing strategies for future research directions it may be a wise decision to reflect on the development in the respective area during the last few years. As to future applications of embedded systems, we consider a concise solution for interconnecting embedded systems to be one of their core requirements. In particular, we focus on the development of dependable communication.
Our paper recapitulates progress in research and development of dependable time-triggered communication protocols as done by the Institute for Computer Engineering at the Vienna University of Technology and by TTTech Computertechnik AG over the last five years. We provide an overview of the current situation and discuss the ongoing research and development directions.

1 Introduction

Dependable communication infrastructures are required in various applications that ensure the standards of our daily life. Such applications range from flight-control systems in aircrafts to distributed control systems in nuclear power plants. The basis of a dependable communication infrastructure is the communication protocol and the properties that it provides. Determinism and predictability are desired properties for communication protocols as they support the reasoning about the system during the development process as well as during the application process of a respective system. Protocols that implement the time-triggered paradigm provide determinism and predictability.

Time-triggered protocols are suitable for the x-by-wire market, for example fly-by-wire in avionics or steer-by-wire in the automotive sector. While time-triggered technology is used in the avionics market for quite some time, it is an emerging technology for the automotive market and is about to hit the market in form of FlexRay^{TM}[Fle05]. Furthermore, as embedded systems are evolving from stand-alone solutions to distributed embedded systems, there is a potential for dependable distributed embedded systems in markets adjacent to the traditional safety-critical ones. Examples of future applications that require dependable communication are distributed implants, human robotics, or novel games. Fault tolerance is a key mechanism in these examples. Assume a distributed implant that consists of a distributed embedded system with sensors and actuators placed in several organs; it shall not happen, e.g., that a “stuck message” [1] causes an overdose of adrenaline. Similar scenarios can easily be constructed for human robotics and novel games, where corrupted communication may lead to significant economic loss, or loss of fun respectively. At a glance, such thinking may appear to be science fiction, and this might be true for the examples above. However, a low-cost dependable communication infrastructure with accompanying novel software technologies is an enabler for such applications.

Time-triggered protocols have been intensively studied over 25 years at the Technical University of Vienna and at TTTech Computertechnik AG since its establishment in 1998. We give an overview on time-triggered communication in Section 2. It is a vision of TTTech to offer a product line of time-triggered protocols that supports a wide range of dependable distributed embedded systems including the traditional safety-critical ones as well as emerging and future systems. One step towards this goal is a layering of protocol services to make it configurable to a customer’s needs. This approach has resulted in the Layered Time-Triggered Protocol (LTTP), a TTP research derivative. We discuss LTTP in Section 3. One particular fault-tolerance mechanism that has been studied intensely over the last years are central guardian instances; we give an overview of the central guardian concepts in Section 4. Ongoing research and development is concerned with bridging the gap between the Ethernet world and dependable time-triggered communication. This research trend is sketched in Section 5. This paper concludes with Section 6.

2 Time-Triggered Communication

In a distributed system where each of the components has access to a local clock, the states of the local clocks can be brought into agreement, that is, the clocks can be *synchronized*. For this purpose there are two types of algorithms: clock-synchronization algorithms and startup algorithms. Clock-synchronization algorithms are used to maintain the quality of the synchronization once a certain threshold is reached. The startup algorithm has to ensure that such a threshold is reached within an upper bound in time. This separation of the synchronization problem into the subproblems of startup and clock synchronization is not always done in the literature and there are clock-synchronization algorithms that solve both subproblems at once. Many of these algorithms, however, either assume a reliable transmission of messages between the nodes *per se* or are of a probabilistic nature.

Furthermore, in a computer system, there is no action that starts *by itself*. An action needs a trigger to be executed. We can distinguish two basic types of triggers: event-triggers and time-triggers. Event-triggers are external triggers that are received by a component either via the communication channels or from the environment. Time-triggers (are triggers that) arise when a clock, to which

[1] This is a message that is continually re-sent by a communication participant, e.g. imposed by a faulty controller.

the component has access to, has reached an action state. These action states can either be defined *a priori*, and be therefore explicitly known to the system's designer, or can evolve from the execution of certain algorithms on a component. An example for an a priori defined action state would be the start of a Task A: *schedule task A at time 12:00*, where *12:00* is the action state of the component's clock. An example for an evolved action state would be the start of a Task B: *schedule Task B after Task A*, where the action state evolves depending on the execution time of Task A.

Synchronization of the local clocks of the components allows action states to be defined throughout the distributed system, such that it is guaranteed that these action states are reached within the precision Π, an off-line calculable parameter. Hence, it is possible to implement synchronized time-triggers, that allow the components to operate as a coordinated whole. Synchronized time-triggers can be used for the communication strategy: we off-line specify the action states when a node is allowed to access the shared medium. If all nodes adhere to this schedule, a fair distribution of bandwidth is guaranteed. Faulty nodes that do not restrict their sending behavior to the specification have to be blocked by additional guardian instances. We call a communication strategy that is based on synchronized time-triggers a time-triggered communication strategy, whereas communication strategies that use unsynchronized (event- or time-) triggers are called event-triggered communication strategies. The communication schedule for time-triggered communication is generated off-line. The time it takes to process through the schedule table once is called a TDMA round (Time-Division Multiple-Access).

A fine property of time-triggered communication is the time-triggered broadcast property that supports agreement algorithms.

From Reliable to Atomic to Time-Triggered Broadcast: A set of processes communicates by exchanging messages and each of these processes produces local output based on the messages exchanged. Informally spoken, reliable broadcast is a mechanism that guarantees that all processes generate the same unordered set of messages as their local outputs.

The broadcast problem introduces two functional primitives: `broadcast()` and `deliver()`. Each process uses the `broadcast()` primitive to distribute messages to all the other processes. Each process uses the `deliver()` function to generate output. Thus, with progress of time, the `deliver()` primitive generates a sequence of messages. A set of processes solves the reliable broadcast problem if it provides [HT94]:

- Validity: if a correct process broadcasts m, it eventually delivers m.
- Agreement: if a correct process delivers m, all correct processes eventually deliver m.
- Integrity: for any message m, every correct process delivers m at most once, and only if m was previously broadcast by a correct sender.

Atomic broadcast is defined as reliable broadcast that fulfills the following additional ordering property:

– Total Order: if correct processes p and q both deliver messages m and m', then p delivers m before m' if and only if q delivers m before m'.

Informally spoken, atomic broadcast guarantees that not only the set of messages is equal within the set of correct processes, but also the delivery order of the messages.

The time-triggered broadcast makes the implementation of the `broadcast()` primitive on a shared medium trivial: each node uses the shared medium in its assigned time slot. Time-triggered broadcast even enhances the atomic broadcast property in that the delivery order of messages is *a priori* known.

3 Layered Time-Triggered Protocol (LTTP)

The prime design goal of LTTP was a clean separation of the communication layer from higher-layer mechanisms. A membership service is for example a higher-layer mechanism. The encapsulation of the communication layer resulted in a more robust protocol state machine including an enhanced fault-tolerant startup algorithm and clique resolution algorithm.

The LTTP protocol distinguishes several protocol phases that can be grouped as follows: the startup phases, which consists of the INIT, INTEGRATION, and COLDSTART phase, the synchronized operation phase, which consists of the SYNC phase, and the external synchronization phase, which consists of the PAUSE_SYNC and the EXTERNAL_STARTUP phase. The phases are depicted in Figure 1.

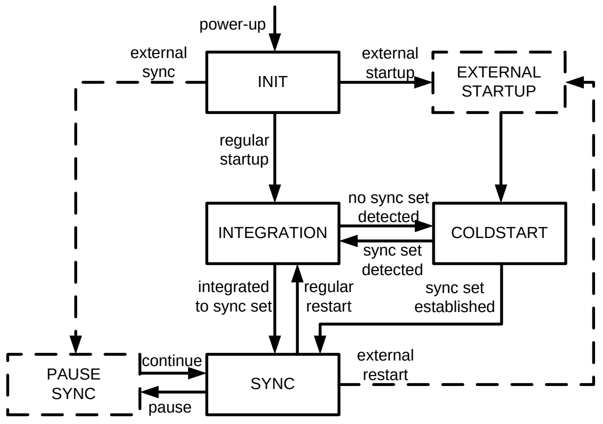

Fig. 1. Protocol Phases

3.1 Protocol Startup

After power-on (that is, after the node is initialized) the node starts the INTEGRATION phase. Each slot in the communication schedule is assigned to a sending node and each message carries the identifier of its sender. Hence, the node listens to the communication channels and has to identify, based on the messages received, if there is a sufficient number of nodes communicating synchronously. If such a set exists, the node integrates into this set and becomes synchronized. If such a sufficient set does not exist, the node enters the COLDSTART phase.

In the COLDSTART phase, the node waits for coldstart signals that are used as starting signal for schedule processing. Such coldstart signals are sent by nodes when a local timer expires (only a subset of nodes may be configured to send a coldstart signal). A node that receives a coldstart signal will start to proceed through the schedule and reply in its assigned slot. The nodes are able to acquire the number of nodes that react to the coldstart signal by counting the replies. The COLDSTART phase ends when a sufficient set of nodes has been synchronized. In general it may also happen that only a subset of nodes in coldstart will reach synchronous operation. For this reason the LTTP startup algorithm defines conditions for a re-transition to the INTEGRATION phase, such that nodes that did not reach the sync phase are able to integrate to the established synchronous communication.

In the SYNC phase the node has reached synchronous operation (but not necessarily steady state). If synchronization is lost, the nodes restart the startup process with the INTEGRATION phase.

The transitions between the different phases of the startup strategy can be taken either by the expiration of timeouts or by the reception of a sufficiently long sequence of messages per TDMA round. It is highly important that a faulty node or channel cannot be able to spread such a sequence of messages (e.g. by masquerading a number of different nodes) that will cause a non-faulty node to take an incorrect transition between startup phases.

To speed up the startup process, LTTP allows using a dedicated TDMA schedule during coldstart. This dedicated schedule may only consist of four slots of minimum size. A more detailed discussion of the startup of time-triggered communication is given in [SK06].

3.2 Synchronized Operation

In LTTP a single node may occupy more than one sending slot in the communication schedule. Furthermore, LTTP introduces the mechanism of "sender-dynamic slots". These slots are scheduled off-line but not statically assigned to a particular node. Instead, the nodes execute an arbitration protocol during run-time to assign the sender-dynamic slots to a particular node. The information for this arbitration protocol is transmitted with the messages in slots that are statically assigned to the nodes. The arbitration protocol used is not part of the LTTP specification; it could be, for example, a sophisticated agreement algorithm to ensure fault-tolerant arbitration or a simple client-server algorithm.

3.3 External Synchronization

LTTP may be used as a sub-bus in a high-speed network. For such and similar purposes LTTP is equipped with provisions to synchronize to external sources. This external synchronization can be achieved in two ways: LTTP can be put into a PAUSE_SYNC phase or into an EXTERNAL_STARTUP phase. In the PAUSE_SYNC phase the communication is halted and operation is resumed either after a given duration or upon an external event. In the EXTERNAL_STARTUP phase LTTP awaits an external startup event upon which the regular coldstart will be executed.

Also, these mechanisms support a global synchronization if several LTTP systems (so called clusters) are connected together to form an LTTP multi-cluster.

3.4 Clique Resolution Algorithms

A potential threat to time-triggered communication protocols is the establishment of a cliques scenario. Cliques are then established, when two sets of disjoint nodes are synchronized within their respective set but the two sets are unsynchronized to each other. In our opinion an assumption that cliques will never form is not acceptable for safety-critical systems as multiple transient failures or faulty communication channels may cause their establishment. Hence, appropriate algorithms to resolve cliques scenarios are required.

TTP uses a so-called clique avoidance algorithm: as the node cyclically proceeds through the communication schedule it classifies each message it receives as correct or incorrect and increases a respective counter. When the node reaches its sending slot in the TDMA round (in TTP each node occupies only one slot per TDMA round) it checks the counters. The node detects cliques when the number of incorrect received messages is higher than the number of correct messages. Hence, this algorithm is based on the relative number of correct messages.

A drawback of this approach is that a node may not receive all incorrect messages, as communication is performed via half-duplex communication links. As a result, depending on the communication schedule configuration there is a probability that cliques scenarios are not diagnosed. In LTTP we propose a clique resolution algorithm that is not based on the relative number of correct messages received, but on the absolute number [SPK06]. If the number of received messages falls beyond an off-line calculable threshold for a given duration, cliques are detected. The threshold is a function of either the number of nodes in the system, or of a dedicated subset of nodes.

3.5 Formal Analysis of (L)TTP Services

Several services, such as the clock-synchronization service and the membership service [Pfe03], have been formally verified by means of theorem proofing using PVS. Rushby gives an overview of formal analysis activities in the time-triggered architecture in [Rus02]. Lately, the TTP and the LTTP startup algorithms have been subject to model-checking studies [SRSP04,SK06] using SAL[2]. Due to the performance of the SAL model checker it was possible to assess the startup algorithms by means of exhaustive failure simulation.

4 The Central Guardian Concept

For the discussion of fault-tolerance methods we define fault-containments regions (FCRs), that are regions that are impacted by a fault and fail as a whole. In distributed embedded networks each node computer and each communication channel forms such an FCR.

When multiple FCRs share a common resource, as in our case a shared broadcast channel, it is necessary to protect that shared resource via additional, independent FCRs. If such a protection mechanism is not implemented, a faulty FCR bears the potential danger to monopolize the shared resource and to render it unusable for other, correct FCRs. Temple introduced the concept of "local guardians" in [Tem99]: a node will not access a shared broadcast channel directly but will communicate with a local guardian which may or may not relay the send attempt to the shared broadcast channel, depending if the local guardian classifies the sending attempt correct or faulty. To tolerate the failure of one local guardian or of one shared broadcast channel itself, the local guardian, as well as the channel, have to be duplicated. This results in a number of $2 * n$ local guardians in a system of n nodes with two replicated channels. To justify the independence argument of FCRs it is required to implement the node and local guardians on separated silicon which makes the local guardian solution economically unattractive. Indeed, the first implementations of the local guardian concept (for TTP) placed the local guardians and the node on the same chip, thus weakening the requirements on a FCR. Fault-injection studies showed that this implementation of local guardians leads to error propagation scenarios [ABST03].

With the movement from a bus topology to a star topology, the promising concept of central guardians was introduced [BFJ$^+$00]: instead of implementing the guardian FCRs locally at the node's side, the guardians are placed at the hubs of the star network. The economic benefit of this solution is obvious, instead of $2*n$ local guardians only two central guardians are necessary in a two-channel system for any number of nodes. The first proof of concept for central guardians [BKS03] basically places a passive node, that is a node without hardware units for message generation, at the hub that executes the same protocol as the regular nodes. The hub controls the dataflow according to the passive node. From a conceptual point of view, this solution is elegant: a node and the central guardian temporally form a self-checking pair, that is, the central guardian is able to transform the arbitrary behavior of a faulty node to a detectably-faulty behavior (with respect to protocol execution). Thus, no semantically faulty messages will pass the central guardian and the fault tree of the system can be kept

[2] PVS and SAL are developed by SRI International.

at a minimum. In particular this first generation of central guardians required following mechanisms:

– the guardian has to execute a semantic filter, that is, certain fields of a messages are analyzed by the guardian and, if a semantic failure is detected, the message is transformed into a syntactically faulty message, by truncation of the message,
– the centralized guardian instances have to use interlinks which are unidirectional direct connections between the two centralized guardians, such that a centralized guardian receives the messages transmitted on the respective other channel, and
– any one non-faulty guardian has to be powered on before any non-faulty node starts to transmit messages.

Another central guardian strategy is a minimum strategy that aims at keeping the state in a central guardian as small as possible. This strategy has certain benefits for reasoning about the fault behavior of the central guardian itself, since we have to argue that even a faulty central guardian will not create valid messages. Such a minimum state strategy for a central guardian was selected for the LTTP protocol. This second generation of central guardians allows dismissing the above listed requirements, although they may be implemented for performance reasons.

5 Dependable Communication on Ethernet

There are several approaches to equip standard Ethernet with real-time capabilities [Fel05]. Probably most notable beyond all is the IEEE activity in form of the IEEE 1588 standard [IEE04]. IEEE 1588 specifies a clock synchronization protocol on top of Ethernet.

An orthogonal approach to IEEE 1588 is Time-Triggered Ethernet (TTE) [KAGS05]. TTE fundamentally distinguishes between foreground time-triggered traffic and background event-triggered traffic, while both traffic classes conform to the Ethernet frame format. Foreground traffic is scheduled *a priori* and prioritized in the TTE switch. A prototype switch [Ste06] developed by the Vienna University of Technology implements a "preemption" mechanism. This mechanism will preempt ongoing transmission of event-triggered messages whenever a time-triggered message has to be relayed. This mechanism provides a high quality on transmission delay and transmission jitter of a time-triggered message. Preempted event-triggered messages will be relayed after the time-triggered message.

However, as IEEE 1588, as well as the synchronization protocol in TTE, are master-slave based protocols, their fault-tolerance capabilities may not be accurate for dependable communication. It is a research activity of TTTech to design a fault-tolerant TTE. One possible solution is to incorporate the (L)TTP services into TTE.

6 Conclusion

Time-triggered technology is successful for dependable communication in well-established markets such as avionics, it is likely to be implemented in emerging markets for dependable communication like, for example, the automotive market, and it is promising and even an enabler for future markets. The LTTP protocol together with the developed guardian instances provide the basis for a robust dependable communication infrastructure and, hence, is suitable to a wide range of applications. On the other side, TTTech is developing Ethernet-based time-triggered protocols with a focus on dependability.

Acknowledgments

Many thanks to the colleagues from the Institute of Computer Engineering, for their hospitality and discussions of many topics. This work was supported by the European Project DECOS (IST-2-511764).

References

ABST03. A. Ademaj, G. Bauer, H. Sivencrona, and J. Torin. Evaluation of fault handling of the time-triggered architecture with bus and star topology. In *Proc. of International Conference on Dependable Systems and Networks (DSN 2003), San Francisco*, Jun. 2003.

BFJ+00. G. Bauer, T. Frenning, A.K. Jonsson, H. Kopetz, and Ch. Temple. A centralized approach for avoiding the babbling-idiot failure in the time-triggered architecture. *ICDSN 2000, New York, NY, USA*, Jun. 2000.

BKS03. G. Bauer, H. Kopetz, and W. Steiner. The central guardian approach to enforce fault isolation in a time-triggered system. In *Proc. of 6th International Symposium on Autonomous Decentralized Systems (ISADS 2003)*, pages 37 –44, Pisa, Italy, April 2003.

Fel05. M. Felser. Real-time ethernet - industry prospective. *Proceedings of the IEEE*, 93(6):1118–1129, 2005.

Fle05. *FlexRay Communications System - Protocol Specification - Version 2.1.* FlexRay Consortium, 2005. Available at http://www.flexray.com.

HT94. Vassos Hadzilacos and Sam Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, 1994.

IEE04. IEEE, INC. *IEEE 1588 – Precision clock synchronization protocol for networked measurement and control systems*, 2004.

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.

Pfe03. Holger Pfeifer. *Formal Analysis of Fault-Tolerant Algorithms in the Time-Triggered Architecture.* PhD thesis, Universität Ulm, Germany, 2003.

Rus02. John Rushby. An Overview of Formal Verification for the Time-Triggered Architecture. In Werner Damm and Ernst-Rüdiger Olderog, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 2469 of *Lecture Notes in Computer Science*, pages 83–105, Oldenburg, Germany, September 2002. Springer-Verlag.

SK06. Wilfried Steiner and Hermann Kopetz. The startup problem in fault-tolerant time-triggered communication. *International Conference on Dependable Systems and Networks (DSN 2006)*, Jun. 2006.

SPK06. Wilfried Steiner, Michael Paulitsch, and Hermann Kopetz. The tta's approach to resilience after transient upsets. *Real-Time Systems*, 32:213–233, Feb. 2006.

SRSP04. Wilfried Steiner, John Rushby, Maria Sorea, and Holger Pfeifer. Model checking a fault-tolerant startup algorithm: From design exploration to exhaustive fault simulation. *The International Conference on Dependable Systems and Networks (DSN 2004)*, Jun. 2004.

Ste06. Klaus Steinhammer. *Design of an FPGA-Based Time-Triggered Ethernet System*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2006.

Tem99. Christopher Temple. *Enforcing Error Containment in Distributed Time-Triggered Systems: The Bus Guardian Approach*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 1999.