TTP – A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems*

H. Kopetz and G. Grünsteidl

Institut für Technische Informatik, Technische Universität Wien
Treitlstr. 3/182/1, A-1040 Vienna, Austria

Abstract

The Time-Triggered Protocol (TTP) is intended for use in distributed real-time control applications that require a high dependability and guaranteed timeliness. It integrates all services that are required in the design of a fault-tolerant real-time system, such as predictable message transmission, message acknowledgment in group communication, clock synchronization, membership, rapid mode changes, redundancy management, and temporary blackout handling. It supports fault-tolerant configurations with replicated nodes and replicated communication channels. TTP provides these services with a small overhead so it can be used efficiently on twisted pair channels as well as on fiber optic networks.

1 Introduction

There exists a growing demand for real-time control systems with guaranteed timeliness and a high dependability. For example, in future computer applications within automobiles [16] critical control information will have to be shared in a timely and reliable manner between autonomous subsystems.

Provided a specified load- and fault hypothesis are maintained, the selected computer architecture must assure a predictable and small bounded maximum latency between a stimulus from and a response to the environment. Furthermore the implementation of fault-tolerance by active redundancy must be supported.

Depending on the triggering mechanism employed, two fundamentally different paradigms for the design of real-time systems can be distinguished: event-triggered (ET) architectures and time-triggered (TT) architectures. In an ET architecture all activities, e.g., task activation, communication, etc., are initiated as a consequence of the occurrence of events, i.e., significant state changes. Because all scheduling and communication decisions in ET architectures are made on-line, they are sometimes called dynamic architectures or interrupt driven architectures [1,7].

TT architectures, on the other hand, are driven by the progression of the global time [15]. All tasks and communication actions are periodic and external state variables are sampled at predetermined points in time. TT architectures are based on stronger regularity assumptions. They are therefore less flexible but are easier to analyze and to test than ET-architectures.

According to the present state of the art, fault-tolerant real-time systems with guaranteed timeliness can only be designed if the base architecture is time-triggered. If the real-time system is event-triggered then it is not known how the problems of replica determinism, systematic testing for timeliness, and timely membership service, can be solved [8].

The Time-Triggered Protocol (TTP) is an integrated communication protocol for time-triggered architectures. It provides the services required for the implementation of a fault-tolerant real-time system, i.e., predictable message transmission, message acknowledgment in group communication, clock synchronization, membership, rapid mode changes, and redundancy management. These services are implemented without extra messages and with only a small overhead in the message size.

In the next section we describe the architectural assumptions and the fault-hypothesis. Section 3 introduces the objectives we had in mind when designing the TTP protocol. Section 4 elaborates on the rationale behind TTP. The detailed protocol description is contained in Section 5. Section 6 discusses protocol characteristics and compares the performance of the TTP protocol with other protocols that have been proposed for control applications.

2 Architectural characteristics

The Time-Triggered Protocol (TTP) is intended for a time-triggered architecture. In such an architecture rapid periodic message exchanges form the bulk of the load. However there is also a need for sporadic event-triggered communication with predictable small latency, e.g., the rapid switchover into another operational mode (e.g., an emergency mode).

2.1 System structure

The distributed computer system consists of a set of fail-silent nodes connected by two replicated broadcast communication channels. To tolerate the failure of a node, nodes can be replicated [22] also and grouped

into Fault Tolerant Units (FTUs). It is guaranteed by the architecture that replicated nodes perform the same state changes at about the same time. As long as at least one node of an FTU is operational, the FTU is considered operational. The real-time clocks of the nodes are synchronized to within a known precision by TTP.

Every node possesses a communication controller with two bidirectional communication ports, connected to the two replicated broadcast channels. Interface nodes have an additional interface to sensors or transducers in the environment. Every node contains error detection mechanisms such that it can terminate its operation in case of an error.

The communication channel is a passive LAN, e.g., a broadcast bus, that transports one message at a time. Access to the communication channel is determined by a time division multiple access (TDMA) schema controlled by the global time generated by TTP. We call a complete cycle, during which each FTU has been granted at least one sending access, a TDMA round.

2.2 Fault hypothesis and system configuration

Failure rate assumptions: We assume that node failures are of the fail-silent type and channel failures are omission failures [5]. The Table 1 gives an order of magnitude of the assumed failure rates.

Failure type	Failure rate $(10^{-9}/h)$
Permanent Node Failure	10^3
Permanent Channel Failure	10^4
Transient Node Failure	10^5
Transient Channel Failure	10^6

Table 1: Failure rates

Although the transient channel failure rate of $10^{-3}/h$ looks high, it is still low if considered on a per message basis. If 1000 messages are transmitted per second, the transient message failure rate is still smaller than 10^{-9} corrupted messages/message. If the independence assumption holds, the probability of two or more message losses occurring within one TDMA round is very low.

During temporary blackout periods the transient failure rates of the nodes and channels can be significantly higher than the failure rates stated above.

System configurations: We call the bit packet that is transported on the physical level a frame. A frame can contain one or more application messages.

We distinguish between the following system configurations that provide increasing levels of fault-tolerance (see Figure 1 and Table 2):

Class I: One node/FTU, 2 frames/FTU. This configuration is relevant if failures in the cabling (e.g. interconnectors, contacts) are dominant.

Class II: Two active nodes/FTU, 2 frames/FTU.

Class III: Two active nodes/FTU, 4 frames/FTU.

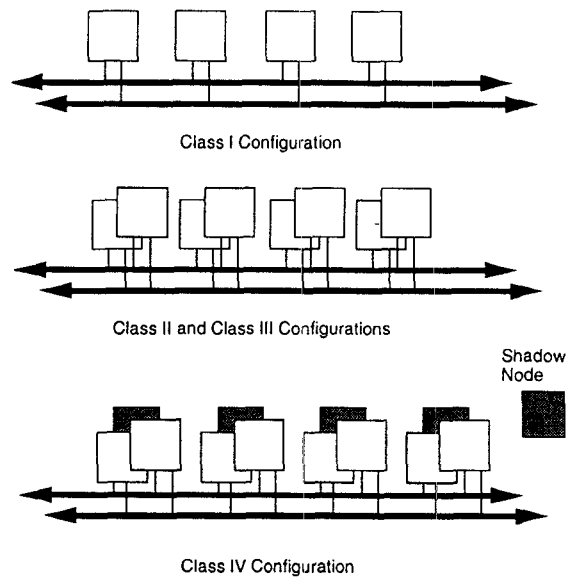*Class IV:* Two active nodes and one shadow node/FTU, 4 frames/FTU.

Figure 1: Fault-tolerant configuration

Tolerance of	Class I	Class II	Class III	Class IV
Perm. Node Fail.	0	1	1	2
Perm. Comm. Fail.	1	1	1	1
Trans. Node Fail.	0	1/Recint	1/Recint	1/TR
Trans. Comm. Fail.	1 of 2	1 of 2	3 of 4	3 of 4

Perm.: Permanent, Trans.: Transient, Fail.: Failure, Comm.: Communication, Recint: Recovery Interval, TR: TDMA Round

Table 2: Level of fault-tolerance

The appropriate system configuration class must be selected on the basis of the intended dependability characteristics and the assumed failure rates in the given application context.

3 Protocol objectives

The objective during the design of the Time-Triggered Protocol (TTP) was to develop an integrated protocol that provides all services needed in a fault-tolerant real-time application. In an automotive context, TTP is intended for class C applications [21] (page 20.272), i.e. for real-time control systems requiring guaranteed timeliness and fault tolerance. The following goals were considered in the design of the protocol:

Message transport with predictable low latency: In a real-time system the temporal accuracy of information is effected by the duration of the protocol execution. A good real-time protocol must have a low maximum execution time and a small variability of the execution time [13] under all specified load and fault conditions. It must handle time-triggered periodic messages and event-triggered sporadic messages.

Fault tolerance: A real-time computer system for safety-critical applications must be fault tolerant. The protocol must tolerate all node and channel failures that

are listed in the fault hypothesis without violating the functional or temporal specification.

Standard communication protocols provide error detection at the sender's site. In real-time applications communication errors that cannot be masked by redundancy must be detected at the receiving site as well as at the sending site with minimal error detection latency.

Temporary blackout handling: A temporary blackout is the temporary interference of some powerful external disturbance with the operation of the control system. The protocol must detect and handle temporary blackouts promptly.

Clock synchronization: The establishment of a global time base with known precision is one of the basic services that must be provided to distributed real-time applications.

Membership service: A membership service provides a consistent view at all nodes about which node is present and which node is absent [4]. In TTP the membership service is the basis for the implementation of atomic multicast protocols and redundancy management protocols. It is also needed to detect incoming and outgoing link failures. Such a failure detection is required for the implementation of the fail-silent abstraction of nodes.

Distributed redundancy management: The removal of failed nodes and the reintegration of spare nodes and repaired nodes has to be controlled by the redundancy management protocol. In a distributed system the redundancy management itself has to be distributed in order to avoid a single point of failure.

Support for rapid mode change: In many real-time applications a set of different operational modes can be distinguished, e.g., start-up, normal operation, emergency, etc.. The protocol should support the consistent and rapid change from one mode to another mode.

Minimal overhead: In many real-time applications, e.g., in automotive electronics, the communication bandwidth is limited. The protocol should provide the specified service with minimal overhead, both in message length and in the number of messages.

Utmost flexibility without compromising predictability: Flexibility and predictability are competing goals. The protocol should provide utmost flexibility as long as the determinism, i.e., the analytical predictability of the timeliness, can be maintained. The protocol should be scalable to high data rates. It should operate efficiently on twisted wires as well as on optical fibers.

4 Design rationale

TTP is an integrated protocol that provides the services listed above without the strict separation of concerns proposed in the layered OSI model. We consider the OSI model an excellent *conceptual* model for reasoning about the different design issues. We do not feel that the OSI model is a good *implementation* model for the development of time-critical protocols, since timeliness was not a goal in the development of the OSI model.

4.1 Sparse time base

Agreement on time, order, membership, and data are fundamental problems in any distributed system. The slightly varying crystal frequencies in the different nodes of a distributed system or the occurrence of faults can lead to major disagreements in the states of replicated nodes if proper agreement protocols are not provided.

In the general case, these agreement protocols are complex and slow [2,17]. If the occurrence of significant events (e.g., the start of sending and receiving of messages) is restricted to the lattice points of a globally synchronized action lattice, i.e., a sparse time-base, then the solutions to these agreement problems are simpler and faster [9]. This restriction simplifies the architecture by reducing the potential input space in the domain of time by orders of magnitude.

TTP is based on such a sparse time base. The granularity of the action lattice g – called 'the basic time granule' – is determined by the precision of the internal clock synchronization. This precision depends on the parameters of the communication channel, the quality of the quartz crystals and the synchronization algorithm. We call the number of bits that can be transported during this basic time granule the bit-length of the basic time granule.

According to our experience the precision that can be expected in the envisioned environment of TTP is less than a few microseconds. On a 250 kbit channel the bit-length of the basic time granule is assumed to be about 2 bits. The introduction of a time-granularity in the microsecond range will have little effect on the macroscopic system properties that are in the millisecond range.

4.2 Use of apriori knowledge

In a time-triggered architecture much information about the behavior of the system, e.g., which node has to send what type of message at a particular point in time of a sparse time base, is known a priori – at design time – to all nodes of the ensemble. TTP tries to make best use of this apriori information. E.g., the message and sender name do not have to be part of the frame if the name of the message can be retrieved from the point in time when the frame is transmitted. Another example relates to error detection. A receiver can detect a missing frame immediately after the apriori known send time has elapsed.

TTP takes advantage of the broadcast facility of the communication medium. It is known apriori that every correct member of the ensemble will hear every frame transmitted by a correct sender. This property of the

broadcast channel is used in the design of the acknowledgment scheme. As soon as one receiver has acknowledged a message from a sender, it can be concluded that the message has been sent correctly and that all correct receivers have received it. To make the acknowledgment scheme fault-tolerant, redundancy is introduced. This line of reasoning is valid as long as the probability of successive asymmetric communication failures is negligible. Experimental evidence from the observation of more than one billion messages has shown that even the occurrence of a single asymmetric communication failure is very unlikely [20].

4.3 State agreement

A receiver can only interpret the frame sent by a sender if sender and receiver agree about the controller state at the time of sending and receiving. In TTP this controller state (the C-state) consists of three fields: the mode, the time and the membership. The mode field contains the identification of the current operational mode of the system. Every operational mode has its own (statically assigned) TDMA sequence, message/frame format, and static task schedule. The time field represents the global internal time. It is also used to denote the position of control within the cyclic mode. The membership field reveals which FTUs have been active and which FTUs have been inactive at their last membership point. The membership points of an FTU are the apriori known points in time when the nodes of this FTU are supposed to send messages [11].

In order to enforce agreement on the C-state without having to include the C-state in each frame, TTP uses an innovative technique of CRC calculation. The CRC is calculated over the frame contents concatenated with the C-state. If the CRC check at the receiver is negative this implies that either the frame has been mutilated or there is a disagreement between the C-states of the sender and receiver. In both cases the frame has to be discarded.

If a particular node did not hear any message from a sending FTU – e.g, because the incoming link of the receiver has failed – it will assume that this sending FTU has crashed and eliminate the sending FTU from its membership. If however all other nodes received at least one of these messages they will come to a different conclusion about the membership. TTP contains a mechanism that makes sure that in such a conflict situation the majority view will win, i.e., that the node with the failed input port, which is in the minority, will be eliminated from the membership. Agreement on membership is thus also tantamount to an indirect acknowledgment of message reception by the majority.

4.4 Fail silence

TTP is based on the assumption that the communication channels have only omission failures and the nodes support the fail-silent abstraction, i.e., they either deliver correct results or no results at all. This helps to en-force error confinement at the system level. If a sender attaches a CRC field to each frame a receiver can detect with a sufficiently high probability whether a frame has been mutilated in the communication channel. If the receiver discards mutilated frames the omission failure abstraction is implemented for the channel.

Designing fail-silent nodes is more complicated. The node implementation must assure by the use of space or time redundancy [12] that all internal failures of a node are detected and the node is turned off before an erroneous output message is transmitted. Moreover, a membership service is required to detect omission failures of the incoming and outgoing communication links.

4.5 Design tradeoffs

In TTP the design tradeoff between processing requirements at the nodes and bandwidth requirements of the channel is tilted towards optimal usage of the available channel bandwidth, even at the expense of increased processing load at the communication controllers. Considering the advances of the VLSI technology it is felt that in many real-time applications, e.g. automotive electronics, the inherent bandwidth limitations [21] (page 20.256) of the channel are much more severe than the limitations in the processing and storage capabilities of the communication controllers.

5 Protocol description

5.1 Controller state (C-state)

As mentioned before, the state of the communication controller (C-state) consists of three fields, a mode field, a time field, and a membership field.

The mode field contains a systemwide unique identification of the current operational mode. Each mode is cyclic, repeating itself after the mode cycle time. The current position within a mode is determined by the value of the current time minus the mode start time modulo the mode cycle time. The following attributes are associated with a mode:

(1) The TDMA sequence of the sending nodes during the TDMA round of this mode.

(2) The attributes (name, format) of each message/frame of each sender at each sending point within the mode.

(3) A static schedule (a dispatcher table) that establishes for every point in the sparse time base modulo the mode cycle time which task has to be executed by each node and which message has to be sent on the communication channel.

(4) A list of successor modes (a succession vector).

The schedule for each mode is developed at compile time. The task dependencies (e.g., mutual exclusion, precedence, etc.) are already considered during the design of a mode schedule [6]. There is no need for dynamic synchronization, e.g., by semaphores. Different modes can contain completely different tasks and different messages.

The time field denotes the current global time. The granularity of the time is a system parameter determined by the basic timing granule (see Section 4.1).

The **membership field** indicates which FTU has been active and which FTU has been inactive at its last membership point. It consists of a bit vector of membership flags, the length of which is equal to the number of FTUs in the present mode.

5.2 Frame format

All information transmitted on the communication channels must be properly framed. A TTP frame consists of the following fields: SOF(start of frame), control field, a data field containing one or more messages, and a CRC field. Between any two frames there is an interframe delimiter.

Start of frame (SOF): The start of frame (SOF) identifies the beginning of a new frame.

Control field: The control field has three subfields: The first subfield with a length of one bit is the initialization bit. It specifies whether the frame is an initialization frame (I-frame) or a normal frame (N-frame). I-frames are needed for the initialization of a node. An I-frame contains the C-state of the sending node in the data field. N-frames are normal data frames containing the application data in the data field.

The next subfield is the mode change field. If it is unequal to zero it specifies an element of the list of successor modes to the current mode.

The third subfield is the acknowledgment field. It contains an acknowledgment for the frames sent by the preceding FTU. The question of which FTU is the preceding FTU depends on the current membership.

Data field: The data field consists of the concatenation of one or more messages containing application data. The length of each message – and thus the data field of each frame – is statically defined in the mode definition. The length of the data fields of separate frames within the same TDMA round can differ, however they must be multiples of a smallest data unit. The length of this smallest data unit is determined by bit-length of the basic time granule. It is not necessary to carry a name field in the frame, since the message name can be inferred from the mode field and the point in time of sending.

Cyclic redundancy check (CRC) field: The CRC field has a length of two bytes. The method for calculating the CRC is described below.

Interframe delimiter (IFD): The Interframe delimiter is required for the proper bit synchronization of sender and receiver.

Typical frame format: Fig. 2 depicts a typical frame format for TTP. The control field has a length of one byte. The mode field (3 bits) allows the specification of 7 successor modes. The length of the acknowledgment field (4 bits) makes it possible to acknowledge each one

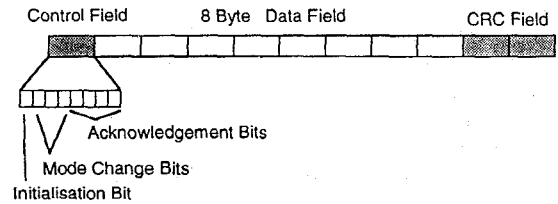of the four frames sent by an FTU in a class III or class IV configuration.

Figure 2: Typical frame format

.... data field in Fig. 2 has a length of 8 bytes. The CRC field has a length of two bytes. The length of the start of frame (SOF) field and the interframe delimiter (IFD) depend on the bus propagation delay, the quality of the clock synchronization, and the employed signalling method. For transmission speeds below 1 Mbit and a bus length below 20 m the SOF length is 1 bit, the IFD length is 3 bits and the basic data unit, determined by the bit length of the basic time granule, is 2 bits. The total frame length for the example in Fig. 2, including SOF, CRC, and IFD is 92 bits. The data efficiency is 69.5%.

5.3 CRC calculation

The 16 bit CRC Calculation conforms with the CCITT standard [3]. This CRC fields makes it possible to detect all single bit errors, all parity errors, and all burst errors less than 17 bits long. For burst errors longer than 16 bits, the CRC misses 0.0015% of the errors.

The CRC calculation is different for Initialization frames (I-frames) and normal frames (N-frames). The first bit of the control field, the initialization field, informs the receiver, whether the frame is an I-frame or an N-frame. In case there is an error in the initialization bit the wrong CRC check will be applied and the frame will normally be discarded. Since it is known apriori for each mode at what points in time I-frames and N-frames will be sent, it is also possible to detect initialization field errors by comparing the actual frame type with the specified frame type.

For I-frames the CRC field is calculated over the control field concatenated with the data field of the frame. They are only accepted if the receiving node is in the start-up phase.

For N-frames the CRC field is calculated over the C-state of the controller (see Section 5.1) concatenated with the control field and the data field of the frame. Since the time, mode, and membership field are typically 2 bytes long, the CRC will detect all errors in any one of these fields with a probability of 100 %. If a message mutilation error coincides with a C-state error – a highly improbable event under the given fault assumption – the detection probability is still 99.9985%.

This kind of CRC checking makes sure, that a normal frame is only accepted if the sender and the receiver have identical C-states, i.e, they agree on mode, time and membership.

5.4 Clock synchronization

TTP provides the fault-tolerant internal synchronization of the local clocks to generate a global time-base of known precision. External clock synchronization is not part of the protocol but can be added by giving a node access to an external time base.

Since the receiver knows a priori the time of sending of each frame, the deviation between the a priori specified send time and the observed receive time is an indication of the clock difference between the sender's clock and the receiver's clock. Thus it is not necessary to carry the value of the send time in the frame.

Continuous clock synchronization is performed without any overhead in frame length or frame number by applying the fault-tolerant average algorithm periodically, preferably with hardware support [14].

5.5 Bus access

Bus access is controlled by the global time. Depending on the fault-tolerance class chosen, a FTU slot will have one or two frame slots. If some senders have to send more information than some other senders, their frame lengths can be different and their sending slot can be repeated more than once in a single TDMA round.

5.6 Membership service

The membership service records which FTU is active and which FTU is inactive at membership points of the nodes (see Section 4.3). We assume that after a FTU has become inactive – it has failed – it will stay in the failed state for at least two TDMA rounds. The join protocol has to guarantee this property.

When an FTU has the right to transmit, i.e., its transmission slot has arrived, it sets its own membership flag in the current membership field to one. At the beginning of the sending slot the receiving nodes set the membership flag of the current transmitter also to one. If no correct frame has arrived from either one of its replicas during the sending slot of a sending FTU, then the receivers set the membership flag of the sending FTU to zero immediately after the sending slot has terminated.

Immediately before its membership point a sending node checks if it is still operating correctly. A node operates correctly

(1) if none of its internal error detection mechanism has indicated that the node is in error and

(2) if at least one of the frames it has sent at its previous sending slot has been acknowledged in the acknowledgments field of at least one of the frames from the succeeding FTU and

(3) if the number of correct frames that it has accepted during the last TDMA round is larger than the number of frames it has rejected because of an unsuccessful CRC check.

The second condition makes sure that the input link and the output link of the node has worked correctly.

The third condition is introduced to avoid the formation of cliques, that is the formation of two or more disjoint subsets of nodes which agree on the C-state within this subset only. If, within a TDMA round, a receiver discards more frames than it accepts it is highly probable that the receiver C-state is in disagreement with the majority of the C-states used in the calculation of the CRC field at the sending nodes. If every sending node sends the same number of messages during a TDMA round, then the receiver has reason to assume that its C-state is in disagreement with the C-states of the majority of senders. As a consequence the receiver enters the inactive state (Fig. 3).

If a node does not operate correctly immediately before its membership point it does not send a frame and becomes inactive. If all nodes of an FTU are inactive, the FTU as a whole is inactive. An inactive FTU will be eliminated from the membership by all correct nodes of the ensemble.

5.7 Mode change

At any point in time, the ensemble of FTUs operates in a particular operating mode. If a FTU intends to change this operating mode, it alerts all other nodes about this mode change by specifying the successor mode in the mode subfield of the control field at its next membership point. In order to reduce the frame overhead, only the position of the successor mode in the statically established succession vector (see Section 5.1) is indicated. Because of this coding mechanism there is no protocol inherent limit in the number of modes that can be supported by TTP.

Mode changes are well-suited to react to sporadic events that require immediate service from the whole ensemble of FTUs. The maximum guaranteed delay interval before a mode change can be activated corresponds to the maximum interval between two sending slots of a FTU. Normally this will be one TDMA round. However if this delay is too long, the respective node can be scheduled more often in the TDMA sequence – with a slot that corresponds to the minimal frame length (without any data field). Such a minimal frame slot is sufficient to activate a mode change, since the mode change field is a subfield of the control field.

5.8 Redundancy management and initialization

Every node has a unique name that refers to its identity. At design time an individual start-up number is assigned to each node. This start-up number determines the position of this node in the TDMA sequence for the cold start.

During the generation of the static message schedules it must be assured that

(1) some nodes send I-frames periodically. The longest interval between two I-frames determines how long a node has to wait until it can be reintegrated.

(2) a node that contains an internal state – the history state or h-state [10] – at its reintegration point will periodically broadcast this state at this reintegration point in order that the partner node can read this reintegration information and resynchronize its internal state . If there is no internal state at the reintegration point, no reintegration message has to be broadcasted.

Fig. 3 depicts the state diagram of a node. When a node is turned on it enters the inactive state. In this state the node performs a self test. After the completion of the self test it enters the start-up state. It resets its local clock to zero and monitors the bus for a prespecified interval I_1 that is longer than the longest TDMA round of all modes.
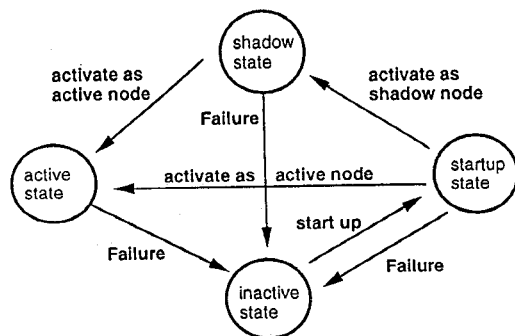

Figure 3: State diagram of a node

If there is no traffic on the bus during this interval I_1 – i.e., the system performs a cold start – the node waits for a further interval I_3 determined by the product of its start-up number and a prespecified start-up interval I_2. This procedure is introduced to reduce the probability of collisions during start-up in case that all nodes of an ensemble are powered up at the same time. After the interval I_3 has elapsed the node sends an Initialization frame (an I-frame) containing its C-state. It repeats this full procedure until it recognizes a frame sent by some other node. As soon as four or more nodes are active, the node switches from the central clock synchronization algorithm to the fault-tolerant average synchronization algorithm.

If there is traffic on the bus during the interval $I_1 + I_3$ the node listens until it receives an I-frame – i.e., the node is reintegrating itself into an operational ensemble. It then updates its C-state with the contents of the received I-frame and participates in the protocol as a listener, updating its C-state regularly. It then waits for the reintegration information of its FTU partner [10]. In the next phase it monitors the traffic on the bus to determine if it is an active node or a shadow node.

If all sending slots in an FTU slot are occupied, then the node is a shadow node. It updates its internal state and performs all calculations without broadcasting any output messages. It monitors the acknowledgment bits of the control fields of the succeeding FTU to determine if one of its FTU partners has failed. If it detects such a

failure it occupies the TDMA slot of the failed partner node and starts sending frames.

After a node has sent a frame it waits for the acknowledgment information in the control field of the succeeding FTU. If none of the frames it has sent is acknowledged in any of the frames from the succeeding FTU it must conclude that it failed and must enter the inactive state. It will then immediately start the reintegration procedure described above. In a class IV configuration with shadow nodes, the shadow partner will have taken the send slot of the failed node in the mean time.

5.9 Temporary blackout handling

A temporary blackout is the correlated failure of a number of nodes because of a powerful external disturbance. Temporary blackout handling requires three phases, blackout detection, blackout monitoring, and recovery from blackout.

Blackout detection: Rapid blackout detection is performed by continuous monitoring of the membership field. If there is a sudden drop in membership – caused be the occurrence of a temporary blackout – then the node performs a mode change to a blackout monitoring mode.

Blackout monitoring: During blackout monitoring a node sends only I-messages and performs – as far as it is still able to do – emergency local control. It will continue to monitor the membership to see if some other nodes start to recover. As soon as the external disturbance disappears the membership will stabilize again. If this is the case a node will initiate the blackout recovery.

Blackout recovery: After the membership has stabilized a mode change to a global emergency service mode will be activated. If during the emergency service another temporary blackout is detected, the node will reenter the blackout monitoring mode. After the emergency service is established, a further mode change to the full service mode will be initialized.

Since TTP monitors the membership continuously, blackout handling can be very quick – in the millisecond range.

6 Protocol characteristics

6.1 Temporal encapsulation of the nodes

The main advantage of TTP over other proposed protocols, such as the contention protocols CAN [21] (page 20.342) and J1850, [21] (page 20.212) or the token protocol [21] (page 20.287), is the temporal encapsulation of the nodes. This temporal encapsulation leads to significant improvements in the fields of testability, system simulation and determinism.

Testability: In many real-time projects more than half of the development resources are spent on testing. The

behavior of real-time systems must be tested in the domains of value and time. This passage focuses on testing the timeliness.

Demand driven protocols, such as CAN, J1850, or token protocols, allocate the communication bandwidth to a particular node dynamically on the basis of this node's current demand. It is hardly possible to determine the peak load interactions of these demands analytically. Therefore, the confidence in the timeliness of an event-triggered system can only be established by extensive system tests on simulated loads. Testing on real loads is not sufficient, because the rare events (e.g., the occurrence of a serious fault in the controlled object), which the system has to handle will not occur frequently enough in an operational environment to gain confidence in the peak load performance of the system. The predictable behavior of the system in rare-event situations is of paramount utility in many real-time applications.

In contrast, TTP allocates the communication bandwidth statically and thus encapsulates every node in the domain of time. Since all uncontrolled interactions between the nodes are prevented, a constructive test methodology can be followed. At first, the temporal behavior of every node is tested in isolation. In a constructive second step the system performance is established. Any discrepancy between the intended and actual temporal behavior at the system level can be localized effortlessly to the offending node.

System simulation: Since the time-base of TTP is sparse and determined by the granularity of the globally synchronized action grid, every input case can be observed and reproduced exactly in the domains of time and value. This property of TTP is important for simulation and debugging. The sparse time-base leads to a significant reduction in the possible number of different execution scenarios that have to be considered in a simulation – as compared to a corresponding ET-system – since in a TT-system the order of the state changes within a granule of time is not relevant [23].

Determinism: The implementation of fault tolerance by active redundancy requires replica determinism – i.e., the replicated nodes must perform the same state changes at about the same point in time. TTP supports the replica determinism of the nodes. Recently, the problems of replica determinism have been investigated extensively [18]. Whereas, the problems of replica determinism in TT architectures have been solved, no equivalent solution is known for real-time ET architectures.

6.2 Flexibility

Extensibility: The TDMA sequence and the data formats of TTP are controlled by the mode definition. If a new FTU is to be added to a given system, new mode definitions have to be generated that include this new FTU. The static scheduler [6] has to check off line if

these new modes will still meet all response time requirements of the given application. If a task in a given mode is modified, it has to be checked statically [19] whether the maximum response time of the new task will still fit in the preallocated execution slot of the old task. If this is the case, this change will have no effect on the timing at the system level. Otherwise a new static schedule for this mode has to-be generated.

TTP allows the design of decomposable systems. Each subsystem can be developed independently and checked against the given specification in the value domain and the time domain. The integration of these independently developed subsystem is straightforward, as the experience with our MARS architecture has shown [10]. This is in drastic contrast to ET-architectures, where every local change in one task can have a global effect on the timing of other tasks in other nodes. The complete regression tests at the system level have to be reexecuted in ET-architectures, after even minor changes in some application task. In an ET-system it is thus hardly possible to determine the proper temporal behavior of a subsystem in isolation.

Compatibility: TTP has a predictable response time and an unrestricted length of the data field. Therefore it is in principle compatible with all protocols that have an unpredictable response time and a restricted data field – such as the automotive protocols in class A and class B [21] (page 20.272). When the information needed by a given protocol is not available in the TTP-control field it is possible to use some bytes of the TTP data field to carry this additional protocol information. The required protocol conversion can be performed locally in the communication controller such that the clients are not aware of the different low level frame formats.

Flexible redundancy: TTP supports different redundant configuration classes. The one byte control field, introduced in Section 5.2, is sufficient to support all of these configurations. If a system based on TTP is properly configured, it is possible to switch from a class I configuration to a class IV configuration (or vice versa) with no changes in the application software – only the replicated hardware resources have to be provided. It is, e.g., possible to start with a class IV configuration in a prototype implementation. If enough information about the failure rates that occur in real life has been gathered, a more economical class III or class I configuration can be implemented in the product market without any change in the application software.

6.3 Performance comparison

It is a delicate issue to relate the performance of communication protocols that are based on disparate architectural paradigms and provide different sets of services. Therefore the following tables have to be interpreted with care. It has to be considered that time-triggered protocols, e.g., TTP, require the same amount of bandwidth independent of the actual de-

531

mand, whereas event-triggered protocols, such as J1850 or CAN, are demand driven. The token protocol lies someplace in between these two extremes. The comparison numbers for the token protocol, J1850 and CAN are taken from [21] (page 20.301). In TTP we assume a class II configuration that will tolerate node and frame losses.

Message length and loop size: Table 3 shows the message length used by the different protocols and Table 4 depicts the comparative loop sizes measured in bits under the assumptions that 8 nodes/FTUs are sending a 16 message loop with 32 total message bytes (2 bytes/message) [21] (page 20.301). In TTP the two messages per node are packed into a frame with a 4 byte data field. The message name can be derived from the point in time of sending. All messages are acknowledged.

Message/Frame Length	Token	J1850	CAN	TTP
SOM	1	1	1	1
ID	16		11	0
Control	0	32	14	4
Data	var.	var.	var.	var.
Ack	16	0-57	2	4
CRC	16	8	16	16
InterFrame	8	3	3	3
overhead with ack	57	var	47	28
overhead without ack	41	43	n.a.	n.a.

Table 3: Message length

	Token	J1850	CAN	TTP
Mess. overhead	656	688	752	224
Token overhead	160	–	–	–
Message data	256	256	256	256
Total loop	1072	944	1008	480
data efficiency	24%	27.1%	25.3%	53.3%

Table 4: Message loop size

Response time: Given a 250 kbit/s channel, the TDMA round of TTP in this example is 1.92 milliseconds in a class I or class II configuration. This is also the worst case delay for a mode switch. If a class III or class IV configuration with 4 replicated messages is selected, the TDMA round is doubled, i.e. just below 4 milliseconds.

The global time-base established by TTP makes it possible to synchronize the point in time of sampling the data with the arrival of the TDMA slot at the sampling node. Therefore the guaranteed data delay of TTP will normally be much less than the full TDMA round. In a well-designed static schedule this data delay can be one or two FTU slots, in the previous example this is in the order of 0.1 msec. If an unsynchronized protocol, e.g. a token protocol, is employed the delay between the sampling point and the transmission point cannot be synchronized and can thus vary by a full Token Rotation Time.

6.4 Implementation considerations

Signalling method: This protocol does not specify the transmission medium or the signalling method. The implementor is free to select the transmission medium best suited for the given purpose. Since TTP is not an arbitration based protocol, there are no restrictions on the signalling method employed. Encoding techniques, such as Modified Frequency Modulation, that have fewer than one transition per bit can be used to increase the channel capacity on twisted pairs. This protocol is also scaling well to high transmission speeds since no bit wise arbitration is required. It is a very efficient protocol for fiber optic systems.

Host interface: The interface between a host computer and the TTP controller can be realized by a dual ported RAM. This RAM contains the control registers for the TTP controller, the descriptor fields of the modes and the memory for the incoming and outgoing data objects. The present global time and the recent history of membership fields are available in special registers.

VLSI implementation: Eventually, TTP has to be implemented at the hardware level in a communication controller. We therefore performed a first order estimate of the hardware complexity of such a controller chip. The two most innovative aspects of TTP are the fault-tolerant clock synchronization and the membership protocol. In the context of the research on the MARS architecture we have implemented a VLSI circuit for clock synchronization (the clock synchronization unit CSU [14]) and have used it experimentally during the past five years. This chip has a transistor count of about 10 000 transistors, including all interface circuitry. We are sure that the clock synchronization in a TTP chip will be simpler. The membership protocol is conceptually unsophisticated. The innovative technique of CRC calculation and the counting of the successful and unsuccessful frame receptions can be implemented in hardware without much effort.

TTP does not use contention mechanism for media access control. It is a conflict free media access protocol which simplifies the interface at the signalling level and makes the protocol scalable to very high transmission speeds, e.g., on optical fiber networks. The other functions of TTP are standard functions found in almost any communication controller. We therefore estimate that the complexity of a TTP controller chip with the two redundant I/O channel is less than 100 000 transistors, excluding the memory. Considering the present level of VLSI integration that is far beyond the 1 million transistor boundary it seems technically possible to integrate a TTP controller into a single chip microcomputer.

7 Conclusion

Distributed real-time control systems must support fault tolerance and must have a guaranteed response

time, even in the specified peak-load and fault scenario. In our research we came to the conclusion that only time-triggered architectures can provide such a predictable service. In this paper we have presented a real-time protocol – the Time-Triggered Protocol TTP – that provides all services that are needed for the implementation of a fault-tolerant real-time system with guaranteed timeliness.

By making best use of the apriori information available in a time-triggered architecture, we have shown that such an integral protocol can be efficient and scalable. The protocol has been implemented and experimentally checked in the context of our MARS project. Concurrently, a detailed reliability model of the protocol is being designed and evaluated.

With the present state of VLSI technology TTP can be implemented as part of a single chip microcomputer. We hope that such an implementation effort can be started in the near future.

References

[1] P. A. Barrett, A. M. Hilborne, P. Verissimo, L. Rodrigues, P. G. Bond, D. T. Seaton, and N. A. Speirs. The DELTA-4 Extra Performance Architecture (XPA). In *Proc. 20th Int. Symp. on Fault-Tolerant Computing*, pp. 481–488, Newcastle upon Tyne, U.K., June 1990.

[2] K. P. Birman and T. A. Joseph. Reliable Communication in the Presence of Failures. *ACM Trans. on Computer Systems*, 5(1):47–76, Feb. 1987.

[3] CCI. *Data Transmission over the Telephone Network: Series V Recommendations, Section V41, The Orange Book VIII.1*. Int. Telephone Union, Geneva, 1977.

[4] F. Cristian. Reaching Agreement on Processor-Group Membership in Distributed Systems. *Distributed Computing*, 6(4):175–187, 1991.

[5] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. In *Proc. 15th Int. Symp. on Fault-Tolerant Computing*, pp. 200–206, June 1985.

[6] G. Fohler. Realizing Changes of Operational Modes with Pre Run-Time Scheduled Hard Real-Time Systems. In *Proc. of the Second Int. Workshop on Responsive Computer Systems*, Saitama, Japan, Oct. 1992.

[7] R. M. Kieckhafer, C. J. Walter, A. M. Finn, and P. M. Thambidural. The MAFT Architecture for Distributed Fault Tolerance. *IEEE Trans. on Computers*, 37(4):398–404, Apr. 1988.

[8] H. Kopetz. Event-Triggered versus Time-Triggered Real-Time Systems. In A. Karshmer and J. Nehmer, editors, *Proc. Int. Workshop on Operating Systems of the 90s and Beyond*, Lecture Notes in Computer Science, vol. 563, pp. 87–101, Springer-Verlag, 1991.

[9] H. Kopetz. Sparse Time versus Dense Time in Distributed Real-Time Systems. In *12th Int. Conf. on Distributed Computing Systems*, pp. 460–467, Yokohama, Japan, June 1992.

[10] H. Kopetz, G. Fohler, G. Grünsteidl, H. Kantz, G. Pospischil, P. Puschner, J. Reisinger, R. Schlatterbeck, W. Schütz, A. Vrchoticky, and R. Zainlinger. Real-Time System Development:- The Programming Model of MARS. In *Proc. Int. Symp. on Autonomous Decentralized Systems*, Kawasaki, Japan, April 1993.

[11] H. Kopetz, G. Grünsteidl, and J. Reisinger. Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System. In A. Avizienis and J. C. Laprie, editors, *Dependable Computing for Critical Applications*, vol. 4 of *Dependable Computing and Fault-Tolerant Systems*, pp. 411–429. Springer-Verlag, 1991.

[12] H. Kopetz, H. Kantz, G. Grünsteidl, P. Puschner, and J. Reisinger. Tolerating Transient Faults in MARS. In *Proc. 20th Int. Symp. on Fault-Tolerant Computing*, pp. 466–473, Newcastle upon Tyne, U.K., June 1990.

[13] H. Kopetz and K. Kim. Temporal Uncertainties in Interactions among Real-Time Objects. In *Proc. 9th Symp. on Reliable Distributed Systems*, pp. 165–174, Huntsville, AL, USA, Oct. 1990.

[14] H. Kopetz and W. Ochsenreiter. Clock Synchronization in Distributed Real-Time Systems. *IEEE Trans. on Computers*, 36(8):933–940, Aug. 1987.

[15] L. Lamport. Using Time instead of Timeout for Fault-Tolerance in Distributed Systems. *ACM Trans. on Programming Languages and Systems*, 6(2):254–280, April 1984.

[16] H.W. Lawson, M. Lidgren, M Stromberg, T. Lundquist, K Lundback, L. Johansson, J. Torin, P. Gunningberg, and H. Hansson. *Guidelines for Basement: A Real-Time Architecture for Automotive Systems*. Mecel, Goteborg, Sweden, May 1992.

[17] L. E. Moser, P. M. Melliar-Smith, and V. Agrawala. Membership Algorithms for Asynchronous Distributed Systems. In *11th Int. Conf. on Distributed Computing Systems*, pp. 480–488, Arlington, USA, May 1991.

[18] D. Powell, editor. *DELTA-4: A Generic Architecture for Dependable Distributed Computing*. Esprit Research Report Project 818/2252. Springer-Verlag, 1991.

[19] P. Puschner and Ch. Koza. Calculating the Maximum Execution Time of Real-Time Programs. *Real-Time Systems*, 1(2):159–176, Sep. 1989.

[20] J. Reisinger. Failure Modes and Failure Characteristics of a TDMA driven Ethernet. Research Report 8/89, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, Oct. 1989.

[21] *SAE Handbook, Vol. 2*. Society of Automotive Engineers, 400 Commonwealth Drive, PA, USA, 1992.

[22] F. B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–319, Dec. 1990.

[23] W. Schütz. On the Testability of Distributed Real-Time Systems. In *Proc. 10th Symp. on Reliable Distributed Systems*, pp. 52–61, Pisa, Italy, Sep. 1991.