TTP — A Protocol for Fault-Tolerant Real-Time Systems

Hermann Kopetz, Technical University of Vienna

Günter Grünsteidl, Alcatel Austria Research Center

The Time-Triggered Protocol integrates such services as predictable message transmission, clock synchronization, membership, mode change, and blackout handling. It also supports replicated nodes and replicated communication channels.

Real-time control systems must share critical information among autonomous subsystems in a timely and reliable manner. For example, automotive applications have separate subsystems for engine and transmission control. Given a specified load and fault hypothesis, the computer architecture must assure a predictable and small bounded maximum latency between a stimulus from and a response to the environment. Furthermore, it must support the implementation of fault tolerance by active redundancy.

Two fundamentally different paradigms for the design of real-time systems are event-triggered architectures and time-triggered architectures (see the sidebar, "Time-triggered real-time architectures"). In an event-triggered architecture, all activities — task activation, communication, and so on — are initiated as consequences of events (significant state changes). Because event-triggered architectures make all scheduling and communication decisions on line, they are sometimes called dynamic architectures or interrupt-driven architectures.

Time-triggered architectures, on the other hand, are driven by the progression of the global time.[1] All tasks and communication actions are periodic, and external state variables are sampled at predefined points in time. Time-triggered architectures are based on stronger regularity assumptions than event-triggered architectures and are therefore less flexible but easier to analyze and test. If the real-time system is time-triggered, then, as we show later, we know how to solve problems of replica determinism, systematic testing for timeliness, and timely membership service.

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

0018-9162/94/$4.00 © 1994 IEEE

In this article we describe our architectural assumptions, fault hypothesis, and objectives for the TTP protocol. After we elaborate on its rationale, we give a detailed protocol description. We also discuss TTP characteristics and compare its performance with that of other protocols proposed for control applications.

Architectural characteristics

Rapid periodic message exchanges form the bulk of the load in a time-triggered architecture. However, a time-triggered architecture also uses sporadic event-triggered communication with predictable small latency — for example, when it rapidly switches to an emergency mode.

System structure. The distributed computer system for which we developed TTP consists of fail-silent nodes connected by two replicated broadcast communication channels. (As we ex-

Time-triggered real-time architectures

Every real-time system has to provide the specified timely service to its environment. To meet this requirement, there are two fundamentally different paradigms: the event-triggered approach and the time-triggered approach.[1]

In an event-triggered system, a significant event in the environment or computer triggers the start of the corresponding system activity — for example, the activation of a task or the transmission of a message. Normally, the interrupt mechanism signals significant events to the operating system. Based on the evolving request scenario, the operating system dynamically determines the order of task executions to meet the specified deadlines.

A time-triggered system initiates all system activities — task activation, message transmission, and the recognition of message arrival — as the real-time clock reaches predetermined values. Ideally, a time-triggered operating system recognizes a single interrupt signal: the ticks generated by the local clock at predetermined points in time. It performs a table lookup to determine the task to activate at each point. A distributed time-triggered system must synchronize the local clocks to represent a globally synchronized time base of specified precision.[2]

Let's compare event-triggered and time-triggered implementations of an elevator-control system in a high-rise building with many parallel elevators. In an event-triggered implementation, each push of an elevator-call button by a client signals the elevator-control system via an interrupt. The control system then immediately decides which elevator is to serve the pending request.

A time-triggered implementation scans the elevator-call buttons periodically, say, every second. It treats all buttons pushed in the last second equally. Based on the result of this scan, the elevator-control system generates a new elevator schedule every second.

In a low-load scenario where a button is pressed only every few seconds, the event-triggered implementation is more responsive than the time-triggered implementation. However, in a high-load scenario where many buttons are pressed during each second, the time-triggered implementation requires less organizational overhead (fewer interrupt service and scheduler calls) and provides better and more predictable service.

With a time-triggered architecture, we know a priori (and design into all nodes) much information about the system's future behavior — for example, which node must send what type of message at a particular time. The TTP protocol we present in this article makes good use of this a priori information. For example, the message and sender name do not have to be part of the frame, since nodes can retrieve this information using the point in time when the frame is sent. Also, for error detection, a receiving node can determine a frame is missing immediately after the anticipated arrival time has elapsed, without the exchange of any acknowledgment information.

The subsystems of a time-triggered architecture are temporally encapsulated by the time-division multiple-access protocol. Since every subsystem in a time-triggered architecture generates its control signals locally from the progression of its local clock, a time-triggered architecture has no control signals crossing subsystem interfaces. Time-triggered systems are therefore more robust and testable than event-triggered systems, but they require more planning and are less flexible. In the elevator application, it is impossible to overload the time-triggered computer system by too many pushes of the call buttons — the flow control is implicit. An event-triggered implementation requires a special mechanism to protect the computer from external overload — the flow control is explicit.

The implementation of active redundancy requires that all replicas visit the same state at about the same time (replica determinism). Since time-triggered systems avoid preemptive dynamic scheduling decisions, they can maintain replica determinism without special agreement protocols.

Safety-critical real-time computer applications for flight control, nuclear power plant shutdown, and so on, have to be fault tolerant and are therefore based on the time-triggered paradigm. For example, the control system for Japan's Shinkansen bullet train uses a large time-triggered fault-tolerant system.[3]

References

1. H. Kopetz, "Should Responsive Systems Be Event-Triggered or Time-Triggered?" *IEICE Trans. on Electronics*, Inst. of Electronics, Information, and Comm. Engineers, Tokyo, Japan, Vol. E76-C, No. 11, Nov. 1993.
2. H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems," *IEEE Trans. Computers*, Vol. 36, No. 8, Aug. 1987, pp. 933-940.
3. A. Hachiga, "The Concepts and Technologies of Dependable and Real-Time Computer Systems for Shinkansen Train Control," *Responsive Computer Systems*, Vol. 7 of *Dependable Computing and Fault-Tolerant Systems*, Springer-Verlag, Berlin, 1993, pp. 225-252.

Table 1. Failure rates.

Failure Type	Failure Rate (10^{-9}/hour)
Permanent node failure	10^3
Permanent channel failure	10^4
Transient node failure	10^5
Transient channel failure	10^6

plain later, fail-silent nodes deliver either correct results or no results at all.) To tolerate node failure, a system engineer can replicate nodes.[2] It can also group nodes into fault-tolerant units (FTUs),[3] which provide one of several levels of fault-tolerant service to their clients. The time-triggered architecture guarantees 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 whole unit is considered operational. TTP synchronizes the nodes' real-time clocks to within a known precision.

Each node has 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. Also, each node has error-detection mechanisms so it can terminate operation in case of an error.

A communication channel is a passive local area network — 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 TTP-generated global time. We call a complete cycle, during which each FTU has been granted at least one sending access, a TDMA round.

Fault hypothesis. We assume that node failures are fail silent and channel failures are omission failures. Table 1 gives the orders of magnitude for the assumed failure rates.

Although the transient channel-failure rate of 0.001 per hour looks high, it is still low if considered on a per-message basis. If the system propagates 1,000 messages per second, the transient message-failure rate is still smaller than 10^{-9} corrupted messages per message. If the independence assumption holds, the probability of two or more message losses within a single TDMA round is very low.

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

System configurations. We call the bit packet transported on the physical level a frame. A frame can contain one or more application messages. Starting with the least fault tolerant, Table 2 presents data for four classes of system configurations:[3]

- Class 1: One node per FTU, two frames per FTU.
- Class 2: Two active nodes per FTU, two frames per FTU.
- Class 3: Two active nodes per FTU, four frames per FTU.
- Class 4: Two active nodes and one shadow node per FTU, four frames per FTU.

Figure 1 shows the configurations. A shadow node in a Class 4 architecture receives all input messages but does not produce any output as long as the FTU's other two nodes are operational. If a node fails, the shadow node takes its right to send and produces output.

We select the system-configuration class for an application on the basis of the dependability we want and the failure rates we assume. For example, a Class 1 configuration may be appropriate if most failures are in the cabling (interconnectors and contacts).

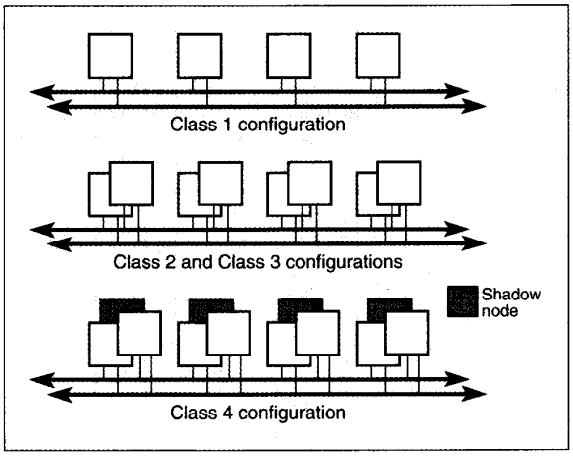

Figure 1. Fault-tolerant configurations.

Protocol objectives

We intended TTP for Class C safety-critical automotive applications:[4] real-time control activities requiring guaranteed timeliness and fault tolerance. Our objective was a protocol that integrated all services needed in such applications:

Table 2. Levels of fault tolerance (in terms of the number of tolerated node failures or frame losses).

Failure Type	Class 1	Class 2	Class 3	Class 4
Permanent node failure	0	1	1	2
Permanent communication failure	1	1	1	1
Transient node failure	0	1/recovery interval	1/recovery interval	1/TDMA round
Transient communication failure	1 of 2	1 of 2	3 of 4	3 of 4

• *Message transport with predictable low latency.* In a real-time system, the duration of the protocol execution affects the temporal accuracy of information. A real-time protocol's execution time must have a low maximum and small variability under all specified load and fault conditions. The protocol must handle time-triggered periodic messages and event-triggered sporadic messages.

• *Fault tolerance.* The protocol should tolerate all transient and permanent node and channel failures listed in the fault hypothesis without violating the functional or temporal specifications. Standard communication protocols provide error detection at the sender's site. In real-time applications, communication errors that cannot be masked by redundancy should 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 the control system's operation by some powerful external disturbance, causing correlated failure of a set of nodes. The protocol should detect and handle temporary blackouts promptly.

• *Clock synchronization.* Because ordinary local clocks are not perfect, the establishment of a global time base with known precision is crucial. When times on local clocks drift apart, the protocol must implement a resynchronization strategy.

• *Membership service.* A membership service gives all nodes consistent information about present and absent nodes.[5] In TTP the membership service is the basis for the implementation of atomic-multicast and redundancy-management protocols. It also detects incoming and outgoing link failures to implement the fail-silent abstraction of nodes.

• *Distributed redundancy management.* A redundancy-management protocol must remove failed nodes and reintegrate spare and repaired nodes. A distributed system distributes the redundancy management itself to avoid a single point of failure.

• *Support for rapid mode change.* Real-time applications have different operational modes: start-up, normal operation, emergency, and so on. The protocol should support rapid change from one mode to another consistently at all nodes.

• *Minimal overhead.* Many real-time applications (for example, automotive electronics) have limited communication bandwidth. The protocol should provide the specified service with minimal overhead, both in message length and number of messages.

• *Utmost flexibility without compromising predictability.* Flexibility and predictability are competing goals. The protocol should provide utmost flexibility while maintaining determinism: the analytical predictability of the timeliness. The protocol should be scalable to high data rates. It should operate efficiently on twisted wires as well as optical fibers, to be scalable to different data rates and to be independent of the transmission medium. The economic benefits of a real-time protocol operating efficiently on twisted wires are very significant for automotive applications.

Design rationale

TTP is an integrated protocol that provides these services without the strict separation of concerns proposed in the layered Open Systems Interconnection model. The OSI model is excellent *conceptually* for reasoning about different design issues. However, timeliness is not one of its goals, and it is inadequate as an implementation model for time-critical protocols.

Use of a priori knowledge. TTP's sparse time base is a globally synchronized time lattice that restricts to lattice points the events under the computer's control — for example, the sending and receiving of messages (see the sidebar, "Sparse time versus dense time"). The time-triggered architecture lets us design many such events into the system a priori.

Sparse time versus dense time

On a *dense time base*, an event can occur at an arbitrary point in time, be observed by more than one node of a distributed real-time system, and be assigned different time stamps — even if all clocks are properly synchronized.[1] The observing nodes avoid confusion by executing an explicit agreement protocol to settle on a single time stamp. However, execution of an agreement protocol requires additional time and communication bandwidth, thus reducing system responsiveness. To avoid this overhead, we restrict event occurrences to proper subsections of the timeline; we call this a *sparse time base.*

We determine the sparseness of TTP's time base, or the *basic time granule*, by the precision of the internal clock synchronization. This in turn depends on the communication channel's parameters, the quartz crystals' quality, and the synchronization algorithm. In the envisioned TTP environment, this precision is on the order of a few microseconds. A time granularity in this range would have little effect on the macroscopic system properties, which are in the millisecond range.

The system can restrict to subsections of the time line only events under its control — for example, the sending of messages or the activations of tasks. It cannot restrict events in the environment (outside the computer) and failure events. In case the same external event is observed by more than one node, say, in a redundant arrangement for fault tolerance, we provide explicit agreement protocols at the interface between the system and its environment.

Because TTP assumes fail-silent nodes that produce correct results (in the value and time domains) or no results at all, a failure event that can occur at any time on a dense time base manifests itself in the time-triggered architecture as an omission failure: The expected message does not arrive at the a priori known time. TTP thus consistently recognizes on the sparse time base all failure events covered by the fault hypothesis, without the need for agreement protocols.

Reference

1. H. Kopetz, "Sparse Time Versus Dense Time in Distributed Real-Time Systems," *12th Int'l Conf. Distributed Computing Systems*, IEEE CS Press, Los Alamitos, Calif., Order No. 2865, 1992, pp. 460-467.

TTP lets us take advantage of the communication medium's broadcast facility. We know a priori that every correct member of the ensemble will "hear" every frame transmitted by a correct sender. Hence, we use a simplified acknowledgment scheme. As soon as one receiver has acknowledged a message from a sender, all those receiving the acknowledgment can conclude that the message has been sent correctly and that all correct receivers have received it. To make the acknowledgment scheme fault tolerant, we introduce redundancy.

This line of reasoning is valid as long as the probability of successive asymmetrical communication failures is negligible. Experimental evidence from the observation of more than 1 billion messages shows that even the occurrence of a single asymmetrical communication failure is very unlikely.

State agreement. A receiver can interpret the frame sent by a sender only 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: mode, time, and membership. The mode field identifies the system's current operational mode. 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 also denotes the position of control within the cyclic mode. The membership field reveals which FTUs have been active and inactive at their last membership point. An FTU's membership points are the a priori known points in time when its nodes are supposed to send messages.

To enforce agreement on the C state without having to include it in each frame, TTP uses an innovative technique for cyclic-redundancy-check calculation: It calculates the CRC over the frame contents concatenated with the C state. If the CRC check at the receiver is negative, either the frame has been mutilated or there is a disagreement between the sender's and receiver's C states. In both cases, the frame is discarded.

If a node does not receive any message from a sending FTU because its incoming link has failed, it assumes that the sending FTU has crashed and eliminates the unit from its membership. If, however, all other nodes of the system received at least one of these messages, they will come to a different conclusion about the membership. In such a conflict, the majority view wins, and the node with the failed input port is eliminated from the membership. Agreement on membership is thus an indirect acknowledgment of message reception by the majority.

A membership service is required to detect omission failures of the incoming and outgoing communication links.

Fail silence. We based TTP on the assumption that the communication channels have only omission failures and the nodes support the fail-silent abstraction: They deliver either correct results or no results at all. This confines errors 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. By using space or time redundancy, the node implementation must ensure that all internal failures are detected and the node is turned off before it transmits an erroneous message. Moreover, a membership service is required to detect omission failures of the incoming and outgoing communication links.

Design trade-offs. We tilted the design trade-off toward optimal usage of the available channel bandwidth, even at the expense of increased processing load at the communication controllers. Considering the advances of VLSI technology, we feel that in real-time applications like automotive electronics, the channel's inherent bandwidth limitations[4] are much more severe than the limitations in the communication controllers' processing and storage capabilities.

Protocol description

Controller state. The state of the communication controller (C state) consists of a mode field, a time field, and a membership field.

The mode 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. Attributes associated with a mode are

- the TDMA sequence of the sending nodes during the TDMA round of this mode;
- the name and format of each message or frame of each sender at each sending point within the mode;
- a static schedule (dispatcher table) to establish, for every point in the sparse time base modulo the mode cycle time, the task that must be executed by each node and the message that must be sent on the communication channel; and
- a list of successor modes (a succession vector).

We consider the task dependencies (mutual exclusion, precedence, and so on) while designing a mode schedule.[6] Each mode's schedule is developed at compile time. There is no need for dynamic synchronization (for example, by semaphores). Different modes can contain completely different tasks and messages.

The time field. The time field denotes the current global time. The granularity of the time is a system parameter.

The membership field. To indicate whether each FTU was active or inactive at its last membership point, the membership field has a bit vector of membership flags, the length of which is equal to the number of FTUs in the present mode.

Frame format. All information transmitted on the communication channels must be properly framed. Between any two frames there is an interframe delimiter to help the sender and receiver

synchronize. A TTP frame has four fields:

• *Start-of-frame field.* The start of frame identifies the beginning of a new frame.

• *Control field.* The control field has three subfields. The first uses one bit to specify whether the frame is an initialization frame or a normal frame. An initialization frame initializes a node and contains in its data field the sending node's C state. Normal frames contain the application data in the data field. The next subfield changes the mode if the field value is unequal to zero and specifies an element of the list of the current mode's successor modes. The third subfield contains an acknowledgment for the frames sent by the preceding FTU. The current membership determines which unit is the preceding FTU.

• *Data field.* The data field consists of the concatenation of one or more messages containing application data. The mode definition statically determines the length of each message and thus the data field of each frame. The lengths of the data fields of separate frames within the same TDMA round can differ. However, they must be multiples of a smallest data unit, whose length is determined by the bit length of the basic time granule. The frame has no name field because the receiver can infer the message name from the mode field and the time of sending.

• *Cyclic-redundancy-check field.* The CRC field has a length of two bytes. We describe the calculation method later.

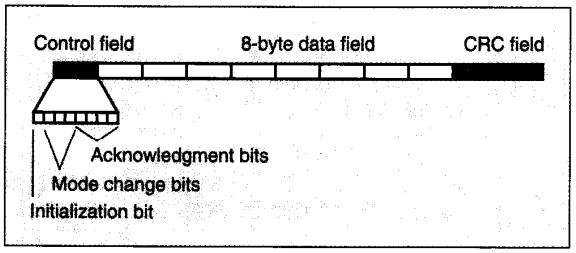

Figure 2. Typical frame format.

Figure 2 shows a typical TTP frame format. The control field has a length of 1 byte. The mode field (3 bits) allows the specification of seven 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 3 or Class 4 configuration.

The lengths of the start-of-frame field and the interframe delimiter depend on the bus-propagation delay, the quality of the clock synchronization, and the signaling method. For transmission speeds below 1 Mbit and a bus length below 20 meters, the start-of-frame field is 1 bit, the interframe delimiter 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 Figure 2, including start-of-frame field, CRC, and interframe delimiter, is 92 bits. The data efficiency is 69.5 percent.

CRC calculation. The 16-bit CRC calculation conforms with the CCITT standard.[7] The CRC fields make it possible for the receiver of the frame to detect all single-bit errors, all parity errors, and all burst errors less than 17 bits long. The CRC misses 0.0015 percent of burst errors longer than 16 bits.

The CRC calculation is different for initialization frames and normal frames. If there is an error in the initialization bit (the first bit of the control field that tells the receiver whether the frame is an initialization frame or a normal frame), the wrong CRC check will be applied and the frame discarded. Since a node knows a priori for each mode at what points in time initialization frames and normal frames will be sent, it can detect initialization-field errors by comparing the actual frame type with the specified frame type.

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

For normal frames, the CRC field is calculated over the C state of the controller concatenated with the control field and the data field of the frame. Since the time, mode, and membership fields are typically 2 bytes long, the CRC will detect all errors in any one of these fields with a probability of 100 percent. 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 percent. Hence, a normal frame is accepted only if the sender and the receiver have identical C states: They agree on mode, time, and membership.

Clock synchronization. In TTP, the fault-tolerant internal synchronization of the local clocks generates a global time base of known precision. External clock synchronization is not part of the protocol, but we can add it 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 specified send time and the observed receive time is an indication of the difference between the sender's and receiver's clocks. Thus, a frame need not carry the send-time value.

TTP performs continuous clock synchronization without any overhead in frame length or frame number by applying the fault-tolerant average algorithm periodically, preferably with hardware support.

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

Membership service. To determine which FTUs are active and which inactive at membership points of the nodes, we assume that after an FTU has failed it will stay in the failed state for at least two TDMA rounds. The join protocol must guarantee this property.

An FTU gains the right to transmit when its transmission slot arrives. It then sets its own membership flag in the current membership field to one. At the beginning of the sending slot, the receiving nodes also set the membership flag of the current transmitter to one. If they receive no correct frame 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 if

(1) none of its internal error-detection mechanisms indicate an error,

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

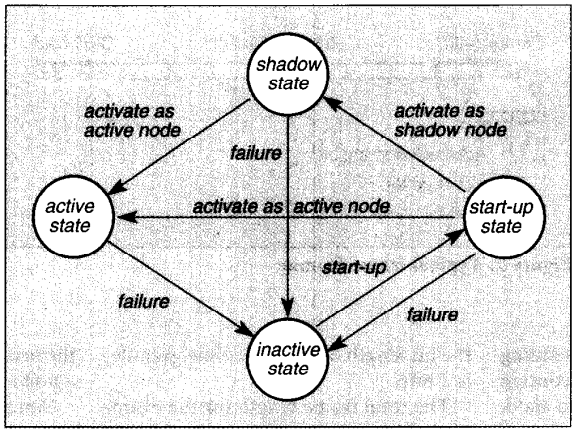

Figure 3. State diagram of a node during start-up and reconfiguration.

The second condition makes sure the node's input and output links have worked correctly. The third condition prevents the formation of cliques — that is, the formation of two or more disjoint subsets of nodes that agree on the C state within their subset only. If, within a TDMA round, a receiver discards more frames than it accepts, it is highly probable that the receiver's C state is not in agreement with the majority of C states the sending nodes use to calculate their CRC fields. If every sending node sends the same number of messages during a TDMA round, the receiver assumes that its C state is in disagreement with the C states of the majority and enters the inactive state.

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. All correct nodes of the ensemble will eliminate an inactive FTU from the membership.

Mode change. At any point, the ensemble of FTUs operates in a particular mode. To change the mode, an FTU alerts all other nodes by specifying the successor mode in the mode subfield of the control field at its next membership point. To reduce the frame overhead, it indicates only the position of the successor mode in the statically established succession vector. With this coding mechanism, there is no protocol-inherent limit to the number of modes TTP can support.

Mode changes let the whole ensemble of FTUs react to sporadic events that require immediate service. The maximum guaranteed delay interval before a mode change can be activated corresponds to the maximum interval between two sending slots of an FTU. Normally this will be one TDMA round. However, if this delay is too long, the FTU that can signal a mode change 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.

Redundancy management and initialization. Each node has a unique name and start-up number to determine its position in the TDMA sequence for the cold start. During the generation of the static message schedules, the system engineer must be assured that some nodes send initialization frames periodically. The longest interval between two initialization frames determines how long a node must wait until it can be reintegrated. Also, a node that contains an internal state — the history state — at its reintegration point must periodically broadcast this state so that the partner node can read it and resynchronize its own internal state. If there is no internal state at the reintegration point, no reintegration message is broadcast.

Figure 3 shows the state diagram of a node. When a node is turned on, it enters the inactive state and performs a self-test. After successful completion of the 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.

If there is no traffic on the bus during this interval I_1, the system is performing 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 reduces the probability of collisions during start-up if all nodes of an ensemble power up at the same time. After the interval I_3 elapses, the node sends an initialization 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 initialization frame. In this case, the node is reintegrating itself into an operational ensemble. It then updates its C state with the contents of the received initialization frame and participates in the protocol as a listener, updating its C state regularly. It then waits for its FTU partner's reintegration information. 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 concludes it has failed and deactivates. It then immediately starts the reintegration procedure already described. In a Class 4 configu-

ration with shadow nodes, the shadow partner takes the failed node's send slot.

Temporary blackout handling. Handling a temporary blackout requires three phases: blackout detection, blackout monitoring, and recovery from blackout. For rapid blackout detection, a node continuously monitors the membership field. If there is a sudden drop in membership — an indication of a temporary blackout — then the node changes to blackout monitoring mode. In this mode, a node sends only initialization messages and performs, as far as it can, emergency local control. It continues to monitor the membership to see if other nodes start to recover. If the external disturbance disappears, the membership stabilizes and the node initiates blackout recovery.

For blackout recovery, the node changes to global emergency-service mode. If during the emergency service a node detects another temporary blackout, it reenters the blackout monitoring mode. If not, it initializes a further change to full-service mode.

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

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

Protocol characteristics

TTP derives its main advantages over other proposed protocols, such as the contention protocols CAN and J1850 or the token protocol,[4] from temporal encapsulation of the nodes. It also offers enough flexibility to make extension or changes in degree of redundancy easy. Concerning performance, TTP's message overhead is minimal, and its response time is short.

Temporal encapsulation of the nodes. TTP achieves significant improvements in testability, system simulation, and determinism through temporal encapsulation of the nodes. In many real-time projects, more than half the development resources are spent on testing. Real-time system behavior must be tested in the domains of value and time. Here we focus on testing the timeliness.

CAN, J1850, and token protocols are demand driven: They allocate the communication bandwidth to a particular node dynamically on the basis of the node's current demand. It is nearly impossible to determine the peak-load interactions of these demands analytically. Therefore, extensive system tests on simulated loads are the only way to establish confidence in the timeliness. Testing on real loads is not sufficient, because rare events like a serious fault in the controlled object will not occur frequently enough in an operational environment to gain confidence in the system's peak-load performance. Predictable system behavior 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. This prevents all uncontrolled interactions between the nodes, letting us follow a constructive test methodology. First, we test the temporal behavior of every node in isolation; then, we establish the system performance. If there is any discrepancy between the intended and actual temporal behavior at the system level, we can easily find the offending node.

Since TTP's time base is sparse and determined by the granularity of the globally synchronized action grid, we can observe and reproduce every input case exactly in the domains of time and value. This significantly reduces the number of different execution scenarios we must simulate: Unlike in an event-triggered system, the order of the state changes within a granule of time is not relevant.[8]

The implementation of fault tolerance by active redundancy requires replica determinism. The replicated nodes must perform the same state changes at about the same time. To accomplish this, TTP guarantees that all correct nodes receive the same messages in the same order at about the same time. It preserves the temporal order of message sending and ensures that these nodes will have the same controller state — that is, the same operational mode, the same membership information, and the same global internal time.

Flexibility. The TDMA sequence and TTP data formats are controlled by the mode definition. If we add a new FTU to a system, we must generate new mode definitions that include it. The static scheduler[6] must check off line whether these new modes will meet all the application's response-time requirements.

If we modify a task in a mode, we must check statically[9] whether the new task's maximum response time will fit in the old task's preallocated execution slot. If it does, this change will have no effect on system-level timing. Otherwise, we generate a new static schedule for this mode.

TTP lets us design decomposable systems. We can develop each subsystem independently and check it against the specification in the value and time domains. Integrating these independently developed subsystems is straightforward, as our experience with the Mars (Maintainable Real-Time System) architecture has shown.[10]

This is in drastic contrast to event-triggered architectures, where every local change in one task can have a global effect on the timing of other tasks in other nodes. After even minor changes in an application task, designers must reexecute complete regression tests at the system level.

Compatibility. TTP has a predictable response time and an unrestricted data-field length. 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 Classes A and B.[4] When the information needed by a protocol is not available in the TTP control field, some bytes of the TTP data field can carry this additional protocol information. The communication controller can perform the required protocol conversion locally, so clients are not aware of the different low-level frame formats.

Flexible redundancy. The 1-byte control field is sufficient to support all the different redundant configuration classes. If a system based on TTP is properly configured, we can switch from a Class 1 to a Class 4 configuration with no changes in the application software — we need only replicate the hardware

Table 3. Message length of varied protocols (in bits).

Message/Frame Length	Token	J1850	CAN	TTP
Start of message	1	1	1	1
ID	16	—	11	0
Control	0	32	14	4
Data	varies	varies	varies	varies
Acknowledgment	16	0-57	2	4
CRC	16	8	16	16
Interframe	8	3	3	3
Overhead with acknowledgment	57	varies	47	28
Overhead without acknowledgment	41	43	n.a.	n.a.

Table 4. Message loop size.

Protocol Characteristics	Token	J1850	CAN	TTP
Message overhead (in bits)	656	688	752	224
Token overhead (in bits)	160	—	—	—
Message data (in bits)	256	256	256	256
Total loop (in bits)	1,072	944	1,008	480
Data efficiency (in percent)	24	27.1	25.3	53.3

resources. Or we could start with a Class 4 configuration in a prototype implementation, gather information about real-life failure rates, and then implement a more economical Class 3 or Class 1 configuration in the product market without any change in the application software.

Performance comparison. Comparing the performance of communication protocols that are based on disparate architectural paradigms and provide different services is a delicate issue. Therefore, the tables we present here must be interpreted carefully. Time-triggered protocols like TTP require the same amount of bandwidth independent of the actual demand, whereas event-triggered protocols, such as J1850 or CAN, are demand driven. The token protocol lies somewhere between these two extremes.

The comparison numbers for the token protocol, J1850, and CAN are from the *SAE Handbook*.[4] For TTP we assume a Class 2 configuration that will tolerate node and frame losses.

Message length and loop size. Table 3 shows the message length used by the different protocols. Table 4 shows the comparative loop sizes measured in bits under the assumption that eight nodes or FTUs send a 16-message loop with 32 total message bytes (2 bytes per message).[4]

In TTP, the two messages per node are packed into a frame with a 4-byte data field. The receiver can derive the message name from the time of sending. All messages are acknowledged.

Response time. Given a 250-Kbps channel, the TDMA round of TTP in this example is 1.92 ms in a Class 1 or Class 2 configuration. This is also the worst case delay for a mode switch. For a Class 3 or Class 4 configuration with four replicated messages, the TDMA round is doubled to just below 4 ms.

TTP's global time base makes it possible to synchronize the time of sampling the data with the arrival of the TDMA slot at the sampling node. Therefore, TTP's guaranteed data delay is normally much less than the full TDMA round. In a well-designed static schedule, this data delay can be one or two FTU slots; for the example considered above, this is on the order of 0.1 ms. With a token protocol, the delay between the sampling point and the transmission point cannot be synchronized and can thus vary by a full token rotation time.

Implementation considerations. To give the implementer freedom to select the transmission medium best suited for an application, this protocol does not specify the medium or signaling method. In fact, there are no restrictions on the signaling method because TTP is not an arbitration-based protocol. An encoding technique such as modified frequency modulation, which has fewer than one transition per bit, can be used to increase the channel capacity on twisted pairs. TTP also scales well to high transmission speeds for fiber-optic systems since it requires no bit-wise arbitration.

We can realize the interface between a host computer and the TTP controller with a dual-ported RAM that 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.

Eventually, TTP must be implemented in a hardware communication controller. Therefore, we made a first-order estimate of such a controller chip's hardware complexity.

The three most innovative aspects of TTP are the fault-tolerant clock synchronization, the membership protocol, and the CRC calculation technique. In our research on the Mars architecture, we implemented a VLSI circuit for clock synchronization (the clock synchronization unit CSU[11]), which we have used experimentally during the past five years. This chip has 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's conflict-free media-access protocol simplifies the interface at the signaling level and makes the protocol scalable to very high transmission speeds. The other TTP functions are standard and found in almost any communication controller. We therefore estimate that the complexity of a TTP controller chip with the two redundant I/O channels is less than 100,000 transistors, excluding the memory. Because the present level of VLSI integration is far beyond

1 million transistors, it seems technically possible to integrate a TTP controller into a single-chip microcomputer.

Only time-triggered architectures can provide the predictable performance needed in distributed real-time control systems. The Time-Triggered Protocol we presented here provides all the necessary services. We have implemented and experimentally checked the protocol on the Mars architecture, and are now designing and evaluating a detailed reliability model. In the near future, we hope to implement TTP as part of a single-chip VLSI microcomputer. ■

Acknowledgments

We acknowledge many useful comments on an earlier version of this work by the Mars (Maintainable Real-Time System) group at the Technical University of Vienna, particularly Johannes Reisinger and Lorenz Lercher. We also thank Kane Kim, Fred Schneider, Lui Sha, and Paulo Verissimo for useful comments on an earlier version of this article. This work was supported in part by Österreichische Nationalbank under project 4128.

References

1. L. Lamport, "Using Time Instead of Timeout for Fault-Tolerance in Distributed Systems," *ACM Trans. Programming Languages and Systems*, Vol. 6, No. 2, Apr. 1984, pp. 254-280.

2. F.B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys*, Vol. 22, No. 4, Dec. 1990, pp. 299-319.

3. H. Kopetz et al., "Tolerating Transient Faults in Mars," *Proc. 20th Int'l Symp. Fault-Tolerant Computing*, IEEE CS Press, Los Alamitos, Calif., Order No. 2051, 1990, pp. 466-473.

4. *SAE Handbook*, Vol. 2, Soc. of Automotive Engineers, Warrendale, Pa., 1992, pp. 20.212, 20.256, 20.272, 20.287, 20.301, 20.342.

5. F. Cristian, "Reaching Agreement on Processor-Group Membership in Distributed Systems," *Distributed Computing*, Springer Int'l, Vol. 6, No. 4, 1991, pp. 175-187.

6. G. Fohler, "Realizing Changes of Operational Modes with Pre Run-Time Scheduled Hard Real-Time Systems," *Responsive Computer Systems*, Vol. 7 of *Dependable Computing and Fault-Tolerant Systems*, Springer-Verlag, Berlin, 1993, pp. 287-300.

7. "Transmission over the Telephone Network: Series V Recommendations," Section V41, *The Orange Book*, Vol. VIII.1, Int'l Telephone Union, Geneva, 1977.

8. W. Schütz, *The Testability of Distributed Real-Time Systems*, Kluwer Academic, Boston, 1993.

9. P. Puschner and C. Koza, "Calculating the Maximum Execution Time of Real-Time Programs," *Real-Time Systems*, Vol. 1, No. 2, Sept. 1989, pp. 159-176.

10. H. Kopetz et al., "Real-Time System Development: The Programming Model of Mars," *Proc. Int'l Symp. Autonomous Decentralized Systems*, IEEE CS Press, Los Alamitos, Calif., Order No. 3125, 1993, pp. 290-299.

11. H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems," *IEEE Trans. Computers*, Vol. 36, No. 8, Aug. 1987, pp. 933-940.

Hermann Kopetz has been a professor of real-time systems at Technische Universität Wien, the Technical University of Vienna, since 1982, and is the chief architect of the Mars project on distributed fault-tolerant real-time systems there. He formerly managed the Computer Process Control Department at Voest Alpine in Austria and was a professor of computer process control at the Technical University of Berlin. His research interests are in real-time systems, fault-tolerant systems, and distributed systems, fields in which he has published more than 70 papers.

Kopetz received his PhD in physics from the University of Vienna in 1968. From 1990 to 1992, he was the chairman of the IEEE Technical Committee on Fault-Tolerant Computing. He was elected an IEEE fellow in 1993 and is also a member of the ACM.

Günter Grünsteidl has been a research engineer in the Control System Architectures Department at the Alcatel Austria Research Center since May 1993. He participated in the Mars project as a teacher and research assistant at the Institut für Technische Informatik at the Technical University of Vienna from 1987 to 1993. His areas of interest include the application of fault-tolerance techniques in real-time systems and distributed algorithms like communication protocols and membership protocols.

Grünsteidl received his Dipl. Ing. in computer science in 1986 and his Dr. Tech. in 1993, both from the Technical University of Vienna. He is a member of the IEEE Computer Society, IEEE, and ACM.

Readers can contact Kopetz at the Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 3/182/1, A-1040 Vienna, Austria, e-mail hk@vmars.tuwien.ac.at; or Grünsteidl at Alcatel Austria Research Center, Ruthnergasse 1-7, A-1210 Vienna, Austria, e-mail g.gruensteidl@aaf.alcatel.at.