A COMPARISON OF CAN AND TTP

H. Kopetz

Technische Universität Wien, Austria
hk@vmars.tuwien.ac.at

Abstract: This paper compares the principles of operation, the services, the dependability mechanisms and the system level properties of distributed real-time systems that are based on the Controller Area Network (CAN) protocol and the Time-Triggered protocol (TTP). The paper comes to the conclusion that CAN is well suited for soft real-time systems where flexibility is important, while TTP is most appropriate for the design of composable hard real-time systems with high dependability requirements.

Keywords: Protocols, CAN, TTP, Real time, Composability

1. INTRODUCTION

The Controller Area Network (CAN) and the Time-Triggered Protocol (TTP) are two fundamentally different communication protocols for the design of distributed real-time systems. CAN belongs to the class of event-triggered protocols where the temporal control signals are derived primarily from non-time events occurring outside or inside the computer system. CAN was originally developed for non-safety critical body- electronics applications within vehicles. TTP belongs to the class of the time-triggered protocols, where the temporal control signals are solely derived from the progression of time. TTP was originally developed for high-dependability hard real-time applications, where timely error detection and fault-tolerance must be provided. For a detailed comparison of soft versus hard real-time systems see (Kopetz 1997, p.12).

It is the objective of this paper to compare these two protocols from a number of different vantage points. The paper starts by describing the structure of a distributed architecture that is shared by both protocols. After discussing the differences between the event-based and state-based view of a system, the principles of operation of CAN and TTP are explained. In the following section the protocol services of CAN and TTP are compared. The dependability enhancing mechanism of CAN and TTP are discussed in Section 5. Section 6 is devoted to system level properties of CAN and TTP based systems, such as composability, extensibility and error containment. The paper closes with a summary and conclusion in Section 7.

2. ARCHITECTURE

The basic architecture of a distributed CAN and TTP based system is alike. A large system is decomposed into a set of subsystems called *clusters*. Each cluster realizes a particular application function. An example for clusters in the automotive context is a cluster for body electronics and a cluster for vehicle dynamics control. A gateway node that is a member of both clusters can implement a data sharing channel between these two clusters.

2.1 Cluster

A *cluster* is a distributed computer system consisting of a set of nodes that are interconnected by a serial channel (see Figure 1). In order to accomplish system functions that cannot be realized on a single node, e.g., the tight coordination of the engine, the steering, and the brakes in the four wheels, the nodes exchange messages via the serial communication channel. Figure 1 depicts an example of a distributed vehicle control system consisting of 7 nodes.

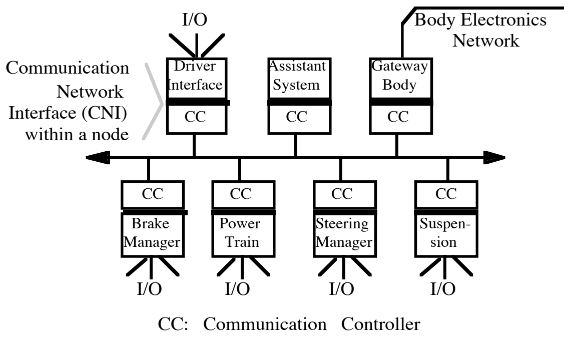

Figure 1: Example of a distributed system consisting of 7 nodes.

2.2 Node

A node of the distributed system consists of three major subsystems, the *host computer*, the *communication controller (CC),* and the *process I/O subsystem* to interface with the signals of the sensors and actuators in the environment (Figure 2). These three subsystems are connected by two interfaces: the *communication network interface* (CNI) between the host computer and the communication controller, and the *controlled object interface* (COI) between the host computer and the process I/O subsystem.

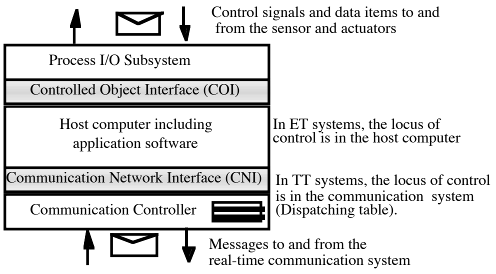
Control signals and data items to and from the sensor and actuators
Process I/O Subsystem
Controlled Object Interface (COI)
Host computer including application software
In ET systems, the locus of control is in the host computer
Communication Network Interface (CNI)
In TT systems, the locus of control is in the communication system (Dispatching table).
Communication Controller
Messages to and from the real-time communication system

Figure 3: Structure of a Node.

Host Computer: The host computer of the node contains a CPU, memory, a local real-time clock, an operating system and the application software. The host computer receives its input data via the CNI and COI and delivers its output data to the CNI and COI. The purpose of the host computer is to execute the computational tasks of the real-time application within the given deadlines.

Communication System: The communication system is formed by the communication channel and the entirety of all communication controllers in the nodes of a cluster. A time-triggered communication controller contains a dispatching table in its local memory that determines at what points in time a particular message is sent or is expected to arrive. An event-triggered communication controller does not need such a dispatching table, because the transmission of a message is triggered by a send command from the host (Figure 3).

The purpose of the communication system is to carry messages from the sender node to one or more receiver nodes in the same cluster within given time constraints. A message carries a statement about attributes of significant state variables (e.g., speed, torque) at a particular point in real-time or about the occurrence of an event. A message is an *atomic unit* consisting of three parts:

(i) the name of the state variable or of the event,

(ii) the observed value of the state variable *v(t)*, and

(iii) the time *t* of observation of the state variable or of the event

Only the value of the state variable/event must be explicitly carried in the message. If the message does not contain the observation time, it is sometimes assumed that the time of arrival of the message at the receiver can be taken as the observation time. In this case the latency jitter, i.e., the difference between the maximum and the minimum protocol execution time, determines the error in the temporal domain.

3. PRINCIPLES OF OPERATION

In this section the principles of operation of a CAN based system and a TTP based system are presented. The section starts with the introduction of the concepts of event messages and state messages.

3.1 Event Messages versus State Messages

To fully appreciate the differences between CAN and TTP it is necessary to gain a full understanding of the differences between event-based and state-based systems. We start with the definition of the basic notions *event* and *state*.

The flow of real time can be modeled by a directed time line that extends from the past into the future. Any occurrence that happens at a cut of this time line is called an *event*. The present point in time, *now,* is a very special event that separates the past from the future. An *interval* on the time line is defined by two events, the *start event* and the *terminating event*. Any property of an object that remains valid during a finite interval is called a *state attribute*, the corresponding information *state information*. An *observation* is an event that records the state of an RT entity at a particular instant, the *point of observation*. A change of state is thus an event that cannot be observed directly, only the consequences of the event, the new state can be observed. The difference between the state information of the state before the event occurrence and the state information of the succeeding new state is called *event information*.

A message is called an *event message* if it is sent immediately after the occurrence of an event and contains in its data field the event information. A message is called a *state message* if it sent periodically at *a priori* known time points and contains in its data field the state information about the observed state. The following Table 1 depicts the differences between state messages and event messages.

Characteristic	Event Message	State Message
Example of message contents	"Valve has closed by 5 degrees"	"Valve stands at 60 degrees"
Contents of data field	event information	state information
Instant of sending	After event occurrence	Periodically at *a priori* known points in time.
Temporal control	Interrupt caused by event occurrence	sampling, caused by the progression of time
Idempotence [Kopetz97, p.110]	no	yes
Handling at receiver	queued and consumed on reading	new version replaces previous version, not consumed on reading
Semantics at receiver	Exactly once	At least once
Consequences of message loss	Loss of state synchronization between sender and receiver	Unavailability of current state information for a sampling interval.
Typical communication protocol	Positive Acknowledgment or Retransmission (PAR)	Unidirectional datagram
Typical communication topology	Point to point	Multicast
Load on communication system	Depends on number of event occurrences	Constant

Table 1: Differences between event messages and state messages.

There are a number of intermediate message forms between state and event messages. For example it is possible--and in many instances useful--to pack the state information about the new state into an event message that is only sent after the occurrence of a state change. An extensive discussion about all these intermediate forms is contained in (Kopetz 1997) p.32.

We call a communication system that is designed for the transmission of event messages an *event-triggered (ET) system*. In an ET system, the signaling of significant events from the environment to the computer system is realized by the well-known interrupt mechanism, which brings the occurrence of a significant event to the attention of the CPU. ET systems require a dynamic scheduling strategy to activate the appropriate software task that services the event.

We call a communication system that is designed for the transmission of state message a *time-triggered (TT) system*. In a time-triggered (TT) system, all activities are initiated by the progression of time. There is only one interrupt in each node of a distributed TT system, the periodic clock interrupt, which partitions the continuum of time into the sequence of equally spaced granules. In a distributed TT real-time system, it is assumed that the clocks of all nodes are synchronized to form a global notion of time, and that every observation of the controlled object is either explicitly or implicitly timestamped with this synchronized time.

3.2 CAN

The specification of CAN used in this paper is taken from the official SAE CAN publication (CAN 1990).

CAN is an event-triggered communication system. Messages are sent if the host computer requests the transmission of a message, the channel is idle and the message priority wins over the messages that other nodes intend to send at about the same time.

Frame Format: CAN supports four different frame types: data frames, remote frames, error frames, and an overload frame. The structure of a data frame is depicted in Figure 3. A data frame consists of 7 fields: the start of frame bit, an 11 bit arbitration field that contains the message identifier, a one bit Remote-Transmission-Request (RTR) field, a 6 bit control field, a data field of between 0 and 64 bits, a 16 bit CRC field, a 2 bit ack field and a 7 bit end-of-frame field.

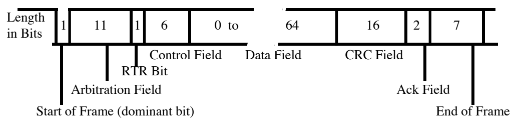

Figure 3: Format of a CAN Data Frame

At the physical layer CAN uses NRZ encoding with bit stuffing. A sequence of more than 5 identical bits is a violation of the bit-stuffing rule. This characteristic feature is used for constructing error frames and overload frames.

Message Arbitration: In CAN, each message type must have a unique identifier that serves two purposes

(i) it assigns a name to the data contained in the message and

(ii) it defines the priority of the message. The identifier with the smallest digital value has the highest priority.

CAN distinguishes between two states of the communication channel: dominant and recessive. The dominant states--which corresponds to the value "0"--overwrites the recessive state. A sending CAN controller must monitor the channel and determine whether its recessive bit has been overwritten by a dominant bit from another sending controller. A losing controller must stop its transmission immediately and continue with the receive operation. The message with the most number of dominant bits--that is the one with the smallest digital value-- will win the arbitration and transmit its data bytes.

Transmission of Messages: The CNI of the CAN controller contains a CAN communication object for every message sent or received. A CAN communication object consists of three descriptor bytes and up to 8 data bytes. The descriptor bytes of a communication object contain the message identifier (11 bits), the data length code (4 bits) and 9 status/control bits.

The host computer requests the transmission of a message by setting the "transmission request" bit in the descriptor byte of the selected communication object in the CNI. The communication controller will start transmission of this message only if the "transmission request" bit is set and the "channel idle" state is detected on the channel. If more than one node intends to transmit a message simultaneously they all will start transmitting the "start of frame" bit as soon as the "channel idle" state is detected on the channel. In the following arbitration phase, where each controller sends the identifier bits contained in the arbitration field (Figure 3), the message with the lowest numerical identifier value will win the arbitration and will be sent on the channel. The receiver stores the data of the message in the data bytes of the communication object at the receiver's CNI. Depending on the setting of the control bits in the receiver's communication object, the receiving controller will raise a "message-received" interrupt to the host computer. After checking the CRC of an incoming message the receiving CAN controller will acknowledge the correct receipt of the message by setting the second ack bit to dominant (Figure 3). The sender thus knows that at least one receiver has received the message correctly.

Remote Frame: The Remote Transmission Request (RTR bit in Figure 3) provides the capability for a node to request the transmission of a particular message from another node. It also serves a diagnostic purpose to determine whether the primary supplier of a given message is functional.

Error Frame: The normal operation of a node is an *error-active* node. If an error-active receiver or the sender detects a transmission error (e.g., incorrect CRC) then an error flag is transmitted by the error detecting node. The error flag consists of six consecutive dominant bits and has thus a characteristic feature that violates the bit stuffing rule. All other nodes will interpret the error flag as a bit-stuffing violation and will transmit their own error flag. The total length of the error flag sequence thus varies between six and twelve dominant bits.

Overload Frame: If a receiving CAN controller requires a delay time before it is ready to receive the next message it can send an overload frame.

3.3 TTP

The specification of TTP used in this paper is taken from the TTP specification that was the basis for the existing TTP implementation produced in the European research projects Brite Euram X-by-Wire and Esprit OMI TTA. Since this implementation is still the subject of evaluation by the industrial partners and the specification may possibly be changed as a consequence of this evaluation, the detailed TTP specification is not in the public domain at the moment.

TTP is a time-triggered communication system. Media-access is controlled by a conflict free TDMA (time-division-multiple access) strategy. Every node is assigned to a unique sending slot in a *TDMA round*. Every TTP controller contains a dispatching table (message descriptor list--MEDL) that contains the information which node is allowed to send which message at a particular point in time. The MEDLs of a cluster are constructed before run time and are *common knowledge* to all nodes. Every TTP controller contains two replicated communication channels in order that the loss of one channel can be tolerated

Structure of the MEDL: The MEDL controls when a message must be sent or received from the communication channels and contains the position of the data in the CNI (Figure 4). The length of MEDL is determined by the length of the *cluster cycle*, i.e., the sequence of TDMA rounds after which the operation of the cluster repeats itself.

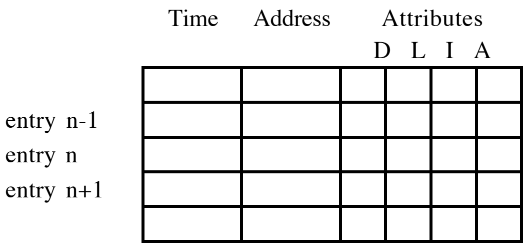

Figure 4: Format of the MEDL.

An entry in the MEDL comprises three fields: a time field, an address field, and an attribute field (Figure 4)**.** The time field contains the point in global time when the message specified in the address field must be communicated. The address field points to the CNI communication objects where the data items must be stored to or retrieved from. The attribute field comprises four subfields:

(i) a direction subfield (D) that specifies if the message is an input message or an output message,

(ii) a length subfield (L), denoting the length of the message that must be communicated,

(iii) an initialization subfield (I) that specifies whether the message is an initialization message or a normal message, and

(iv) an additional parameter subfield (A) that contains additional protective information concerning mode changes.

The host can only execute mode changes that are permitted by the attribute field of the MEDL. In a safety-critical system, all mode changes requested by a host can be blocked by the MEDL. The host cannot access the MEDL that is stored in the TTP communication controller.

Frame Format: TTP supports two different frame types: normal (N) frames and initialization (I) frames. The structure of an N-frame is depicted in Figure 5. A normal frame consists of 4 fields: the start of frame field, a 4 bit header field, a data field of between 0 and 128 bits, and a 16 bit CRC field. The first bit of the header field informs whether the frame is an I-frame or an N-frame. The other three header bits are used to activate a mode change. During normal operation, these three mode-change bits are not set. An I-frame has the same format as an N-frame but contains in its data field the *controller (C) state* of the TTP controller. This C-state consists of the global time, the index of the current MEDL entry, and the membership vector. The membership vector is a bit vector where every node of the system is assigned to an *a priori* specified bit position. This bit is set to "TRUE" if this node has sent a correct message in the previous TDMA round, otherwise it is set to "FALSE". I-frames are required for the initialization of a TTP based system and for the reintegration of failed nodes.

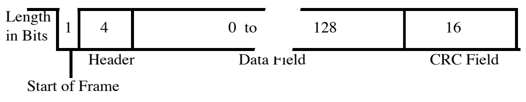

Figure 5: Format of a TTP Frame

At the physical layer TTP uses MFM coding (Miesterfeld 1991) because MFM coding supports bit synchronization without a data-dependent frame length.

Transmission of Messages: The CNI of the TTP controller contains a communication object for all relevant messages sent or received. A TTP communication object consists of one descriptor byte and up to 16 data bytes. The descriptor byte of a communication object contains the message receive status (4 bits), and a concurrency control field (4 bits) for the Non-Blocking-Write (NBW) Protocol (Kopetz and Reisinger 1993). The NBW protocol is used for concurrency control when the host computer and the communication controller intend to access a communication object at the same time.

The communication controller will send a message whenever the global time reaches a value that is equal to the value for this message send time in the (controller internal) MEDL. The message is sent on both channels. If one of the two messages arrives correctly at the receiver, the receiver will set the membership bit of the sender to "TRUE" and store the message in the communication object of the receiver, as indicated in the receiver's MEDL. In case none of the two messages arrives correctly at the receiver, the sender will set the membership bit of the sender as "FALSE". The missing message is marked in the message receive status field of the receiver's communication object.

Initialization Frame: During MEDL design it must be ensured that some nodes send periodically I-frames with the internal state (C-state) of the sending TTP controller. A reintegrating node synchronizes itself with the rest of the cluster by setting its internal state to the C-state received in the I-frame.

Continuous State Agreement: To enforce agreement of the controller states (C-state) among the ensemble without having to include the C-state in each message, TTP uses an innovative technique of CRC calculation for N-Messages (Figure 6). The CRC at the sender is calculated over the message contents concatenated with the sender's C-state. The CRC check at the receiver is calculated over the received message contents concatenated with the receiver's C-state. If the result of the CRC check at the receiver is negative, then, either the message has been corrupted during transmission or there is a disagreement between the C-states of the sender and receiver. In both cases, the message must be discarded.

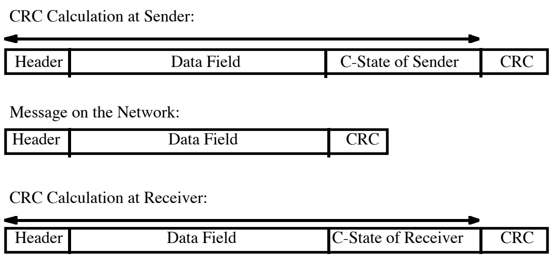

Figure 6: Calculation of the CRC of normal messages.

4. PROTOCOL SERVICES

4.1 Data Integrity

Both protocols, CAN and TTP protect the data field of a message by a 16 bit CRC field. In TTP the CRC has the additional function to detect a C-state disagreement between sender and receiver. The normal Hamming distance in both protocols is 6.

4.2 Throughput

CAN: The arbitration logic of CAN requires bit synchronization, i.e., that every bitcell must stabilize on the channel before the next bitcell can be transmitted. This limits the maximum throughput of CAN to about 1 Mbit/sec in a network of about 100 m length. The assymetric states on the physical communication channel--a high resistance recessive state and a low resistance dominant state--lead to an asymmetric analog waveform of the channel.

TTP: TTP requires message synchronization, but no bit synchronization. There is no protocol limit to the throughput rate of TTP based systems. The prototype TTP controller chip under construction will support transmission rates of 512 kbits/sec, 1 Mbit/sec, and 2 Mbits/sec on a twisted pair. TTP based systems are well suited for implementation on high-speed fiber networks.

4.3 Latency

The time interval between the start of transmission of a message at the sender's CNI and the delivery of the message at the receiver's CNI is called the *latency* of a message. The variability of this delay, i.e., the interval d_{max}-d_{min}, where d_{max} is the maximum latency and d_{min} is the minimum latency is called the *jitter* of the communication system.

Control applications are sensitive to a large latency (increase of the dead time) and even more sensitive to jitter. In the SAE class C application requirements document SAE J2056/1 (SAE 1995) it is stated on p. 23.441: "*From the application perspective, latency relates to the time delay associated with the transfer of information from one application program to another. In this respect it is necessary for the latency to be minimal (in some cases less than 1 millisecond) and predictable (as defined by the systems exchanging data). Predictability has to do with how the message latencies change over time--whether they follow a statistical distribution or are deterministic. As a control system is designed, the delay resulting from information transfer is accounted for in such a way that the control algorithm is able compensate for that fact. From this perspective the necessity of predictable latency (latency defined within a given tolerance) can be understood.*" In control application the jitter should be two orders of magnitude smaller than the latency.

An application that will not work properly if the jitter is beyond a given application- specific value is called a *jitter-sensitive application*. Excessive jitter can be the cause of two types of errors in a jitter-sensitive real-time application: it can give rise to the sporadic creation of orphans and it will introduce an additional measurement error.

Orphans: Consider a scenario where a client requests a service from a server. In order to detect a failure of the server, the client activates a client time-out concurrently to sending the request message to the server (Figure 7).

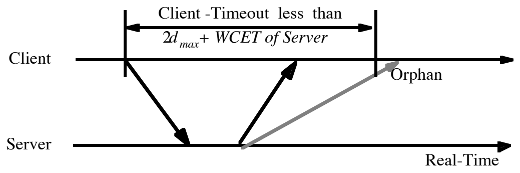

Figure 7: The creation of orphans by an incorrect value of the client time-out.

If this client time-out is less than the maximum reply interval from the server (which is two times the maximum transmission latency plus the worst-case execution time at the server) then a reply message will arrive after the client has started the time-out processing. This late reply message is called an *orphan*. If the orphan is not properly handled, it can cause a transient failure of the application.

The probability of an orphan occurring depends on the form of the latency distribution. Consider a typical latency distribution as shown in Figure 8. The probability that the latency will be larger than an application specific critical latency value, e.g., the one determined by the size of the client time-out parameter in Figure 7, is rather small, giving rise to sporadic difficult-to-reproduce system failures.

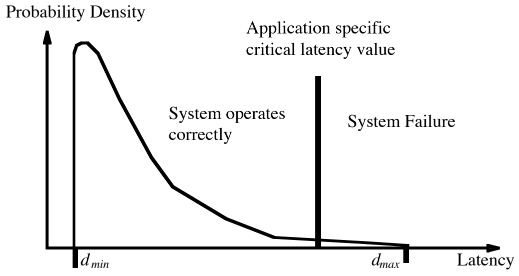

Figure 8: Typical probability density distribution of the latency

Additional Measurement Error: If there is no global notion of time available in a distributed control system, a *jitter* Δt of the communication system introduces an additional measurement error of the size

$$\Delta v = \frac{dv(t)}{dt} \Delta t$$

where *v(t)* denotes the time variability of the measured value *v*. It depends on the characteristics of the given application whether this additional measurement error, introduced by the jitter of the communication system, is of concern. In control applications this additional measurement error is often interpreted as an additional perturbance of the control loop and causes a degradation of the control quality.

CAN: In CAN the occurrences of events in the host computer (the execution of the "*send message command*" in the host software) determine when the communication system must send a message. The jitter of the highest priority CAN message is bound by the longest message transmission interval. If the host of a node sends continuously highest priority messages (e.g., because of a babbling idiot error in the application software of the host computer), then no other nodes will be able to access the channel at all. Thus a number of assumptions about node behavior must be made to determine the worst-case jitter of non-highest priority CAN messages. For a complete analysis of hard real-time communications the reader is referred to the excellent contribution by (Tindell 1995 et al.).

In a system without a global time the host computers operate asynchronously at their own pace. It is not possible to control *a priori* the phase relationships between the execution of the "*send message commands*" in the different nodes of Figure 1. Assume that every node of Figure 1 sends a sporadic message with a minimum interarrival time of *1* msec. Assume further that all messages are of the same size, and it takes *100* µsec to transmit a message. We define a *critical instant* (Liu and Layland 1973) as a point in time when all nodes connected to the channel intend to send a message simultaneously. Since the occurrence of a critical instant, where all 7 nodes execute the "*send message command*" at the same time, cannot be ruled out, the worst case jitter, in the above example, is *600* µsec. The actual latency in the above example will vary between the minimum of *100* µsec (channel idle) and the maximum of *700* µsec (critical instant) depending dynamically on the load generated by the hosts.

In general, the latency in a CAN based system depends on the maximum number and the activity of nodes on the channel. This makes it difficult to select a proper value for the client time-out in the application software of the client at design time. There is a further dilemma: To avoid orphans, the client time-out must be set to the largest latency value that is ever expected to occur. However the client time-out determines the error detection latency that should be small in safety critical systems (see example in Section 4.4 below).

TTP: In TTP every host knows a priori when a message will be transmitted. It can schedule the host local activities (e.g., sampling of the environment) using this common knowledge. The latency jitter in TTP is determined by the precision of the clock synchronization, i.e., in the order of microseconds.

4.4 Membership

In dependable systems, a failure of a node must be reported in a consistent manner to all operating nodes within a low latency. This is the task of the *membership service*. A point in real-time when the membership of a node can be established, is called a *membership point* of the node. A small temporal delay between the membership point of a node and the instant when all other nodes of the ensemble are informed in a consistent manner about the membership, is critical for the correct operation of many safety-relevant applications.

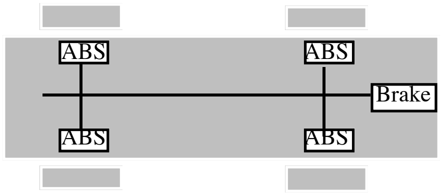

Figure 9: Example of an intelligent ABS in a car.

Example: Consider an intelligent ABS (Antiblock System) braking system in a car, where a node of a distributed computer system is placed at each wheel. A distributed control algorithm in each of the four wheel computers calculates the brake-force distribution to the wheels (Figure 9), depending on the position of the brake pedal actuated by the driver. If a wheel computer fails or the communication to a wheel computer is lost, the hydraulic brake-force actuator at this wheel autonomously transits to a defined state, e.g., in which the wheel is free-running. If the other nodes learn about the computer failure at this wheel within a short latency, e.g., a single control loop cycle of about 5 msec, then the brake force can be redistributed to the three functioning wheels, and the car can still be controlled. If, however, the loss of a node is not recognized with such a low latency, then, the brake force distribution to the wheels, based on the assumptions that all four wheel computers are operational, is wrong and the car might go out of control.

CAN: In a purely event-triggered systems, such as CAN based systems, it is impossible to implement a timely membership service at the protocol level. If a node does not receive a message from another node within a given time interval it cannot distinguish between the case when the node has failed from the case when there was no event occurrence at the sender's node (and therefore the operational sender did not send a message). In CAN the membership service must be implemented in the host computer by executing a periodic time-triggered task in the operating system that sends special membership messages to all other nodes.

TTP: TTP provides a timely node membership service as part of the protocol. The number of bits in the membership field of the C-state corresponds to the maximum number of nodes in a cluster. Every node-send slot is a membership point for the sending node. If one out of the redundant messages of the sending node is correctly received by a receiving node, the receiving node considers the sending node operational at this membership point. The node is considered operational until its following membership point in the next TDMA round. If a node fails within this interval, the failure will only be recognized at the coming membership point. The delay of the membership information is in the order of one TDMA round.

If a particular node did not receive any correct message from a sending node–e.g., because the incoming link of the receiver has failed–it assumes that this sending node has crashed, and it eliminates the sending node from its membership vector. If, however, all other nodes received at least one of these messages they come to a different conclusion about the membership. From this moment onward, two cliques have formed that cannot communicate with each other because they contain a different C-state. TTP contains a mechanism that makes sure that in such a conflict situation the majority view wins, i.e., that the node with the failed input port, which is in the minority, is eliminated from the membership.

4.5 Clock Synchronization

A global notion of time is required if time stamps that are generated locally at one node must be interpreted at another node.

CAN: The CAN protocol specification does not include a clock synchronization service. It is possible to implement distributed clock synchronization at the host level. This requires the transmission of additional high priority messages with low jitter.

TTP: TTP provides the fault-tolerant internal synchronization of the local clocks to generate a global time-base of known precision in the microsecond range. Because every receiving node knows *a priori* the expected time of arrival of each message, the deviation between the *a priori* specified arrival time (contained in the receiver's MEDL) and the observed arrival time is an indication of the clock difference between the sender's clock and the receiver's clock. It is thus not necessary to exchange explicit synchronization messages or to carry the value of the send time in the message, thus extending the message length. Continuous clock synchronization is performed without any overhead in message length or message number by periodically applying a fault-tolerant clock synchronization algorithm, i.e., the FTA algorithm (Kopetz and Ochsenreiter 1987) with hardware support from the TTP Controller.

5. DEPENDABILITY

There are fundamental differences in the support for dependability between CAN and TTP based systems.

5.1 Error Handling Strategy

CAN: The principle error-handling strategy of CAN is an immediate retry. If a communication error is detected in a CAN based system, the error-detecting node sends an error frame and requests the retransmission of the erroneous frame. Any error-active node of a CAN based system can request such a retransmission of a message. To reduce the probability that a single erroneous node interferes continuously with the proper operation of the CAN network, every CAN controller maintains local error counters. If the error counters increase beyond a preset limit--indicating that the node itself is the cause of the observed error--then the node will enter the error passive state.

TTP: The principle error-handling strategy of TTP is error masking, fail silence and restart after a self test. Two copies of every message are sent, one on each replicated channel. As long as any one of these two messages arrives, the timely service is provided. If a node failure must be tolerated, it is assumed that in a fault-tolerant reconfiguration a self-checking replicated node will continue to provide the requested service. In this scenario, four physical messages are sent for every logical message. If a fatal communication error is detected in a TTP based system the error detecting node records this error in its local membership vector and does not change its temporal behavior. If the error is transient, a correct message will be received in the next TDMA round. If the detecting node is at fault (e.g., because of an incoming link failure), the clique avoidance algorithm of TTP will force a fail-silent shutdown of the erroneous node.

5.2 Babbling Idiot Avoidance

A *babbling idiot* is a node that tries to send messages at incorrect points in time. A babbling idiot can interfere with the correctly operating nodes and thus cause the total loss of communication on a single bus. Babbling idiot failures are considered to be the most critical failures in a shared channel system.

CAN: CAN does not provide any special mechanism to handle babbling idiot failures.

TTP: TTP contains a number of mechanisms to avoid the total loss of communication by babbling idiots. At the physical layer TTP supports replicated channels. At the next level up, the TTP controller contains an independent device, the *bus guardian* (driven by its own crystal resonator), to ensure that a node will only transmit during its statically preallocated time slot. The sending activity of the node is controlled by the controller internal MEDL data structure that is not accessible to the host of the node. Therefore even a maliciously faulty host cannot interfere with the proper temporal operation of the communication system in a correctly configured TTP cluster.

5.3 Replica Determinism

A set of nodes is *replica determinate*, if all the nodes in this set contain the same externally visible state, and produce the same output messages in the same order at points in time that are at most an application specific time interval apart. Replica determinism is needed to:

(i) Implement fault-tolerance by active redundancy (Schneider 1990): If the replicated nodes proceed along significantly different computational trajectories, then, the switchover from the result of one replica to that of the other will upset the controlled object, and may even lead to a serious error. The voter in a fault-tolerant system based on majority voting may reach an erroneous result if the inputs to the voter are not replica determinate.

(ii) Facilitate the system test: A replica deterministic system always produces identical results, in the value domain and the time domain, from the same input data presented at exactly the same relative points in time. A non-determinate system may produce different results from identical input data, thus complicating the regression test and the debugging of the system.

The basic causes of replica non-determinism are: differing inputs, different order of messages, a difference between the progress of the computation and that of the local clocks in the replicas, differing oscillator drifts caused by the physical variations of the resonators, and algorithmic peculiarities. For a further discussion of the deep issues of replica determinism the reader is referred to the book (Poledna 1995).

CAN: CAN is not a replica deterministic communication system. It is not possible to guarantee that the message order on two replicated CAN channels of a single node will always be the same.

TTP: TTP is is a replica deterministic communication system based on a sparse time-base (Kopetz 1997, p.57). It is up to the software in the host of a node to provide replica determinism at the host level. It is thus possible to build fault-tolerant systems by implementing actively redundant replica deterministic nodes connected by a TTP communication system.

6. SYSTEM LEVEL PROPERTIES

6.1 Composability

A system is said to be *composable* with respect to a specified property if the system integration will not invalidate this property once the property has been established at the subsystem level. Examples of such properties are timeliness or testability. In a distributed real-time system, the integration effect is achieved by interactions among the different nodes. Therefore, the communication system plays a central role in determining the composability of a distributed architecture with respect to its temporal properties.

CAN: In a CAN based system the temporal properties of the data in the CNIs, e.g., the time it takes for sending the data from one CNI to another CNI, are not determined within the communication system alone. It is within the sphere of control of the host computers (see Figure 2) to decide when messages are to be sent and what channel access conficts will occur. Temporal control in a CAN based system is thus a global issue, depending on the behavior of the application software in all nodes of the distributed system. It follows that the temporal properties of the data in the CNIs are modified during the integration of a system. Only if the software of all nodes adheres to a predetermined access pattern it is possible to give bounds for the maximum latency. The latency jitter is in the same order of magnitude as the latency. A CAN based system is thus not composable with respect to its temporal properties.

TTP: In TTP the temporal control of the communication network is determined by the contents of the MEDLs in the communication controllers, and is not dependent on the application software in the host. The temporal properties of the data in the CNIs--such as when a new version of a message will arrive and how long it takes to transmit a message-- are precisely specified at design time. It is thus possible to test each host individually with respect to its precisely specified CNI. Since system integration will not change these temporal properties of the CNI, a TTP based system is composable with respect to its temporal properties.

6.2 Extensibility

Extensibility refers to the effort required to extend an operational system, by adding new nodes or new messages.

CAN: In CAN a new active node can be added by simply connecting it to the CAN bus. Provided the applications are not jitter-sensitive, the extended network will operate with minimal additional software effort. In a jitter-sensitive application, the difficult-to-predict change in the worst case latency caused by the addition of the new node can be the reason for sporadic timing failures.

Listening error-inactive CAN nodes can be added to a CAN network without any change in the temporal control pattern of the CAN network.

TTP: In TTP a new node can only be added simply if a sending slot for the new node has been reserved during the design of the controller internal MEDLs. If such a change has been prepared for in the design of the MEDL the addition of a new node will not change the temporal properties of any of the CNIs. Otherwise, all MEDLs must be modified to accommodate the sending slot of the new node. Such a MEDL change may lead to an alteration in the temporal properties of the CNIs, the consequences of which can and should be investigated before the change is made.

Listening TTP nodes can be added to a TTP network without any change in the temporal control pattern of the TTP network.

6.3 Error Containment

A large dependable computer system must be structured into partitions that act as *error-containment regions* in such a way that the consequences of faults that occur in one of these partitions can be detected and corrected or masked before these consequences corrupt the rest of the system. Well-defined error containment regions simplify the diagnosis of faults and are a necessary prerequisite for the certification of large systems consisting of critical and non-critical components.

In a distributed system it is conducive to consider a node with the system- and application software as an integral unit of error containment. The interfaces between a node and the rest of the system, i.e. the CNIs, must support a high degree of error detection such that an error that occurs inside the host of a node does not propagate across the CNI to the rest of the system.

CAN: CAN does not support any mechanism of error containment across the CNI. A single error (software or hardware) in any host connected to a CAN channel can cause the total loss of communiction between all nodes on this channel.

Characteristic	CAN	TTP
Application domain	Soft real-time systems with flexibility requirements.	Hard real-time systems with composability, timeliness and dependability requirements.
Temporal control	Event triggered from host	Time triggered within the communication system
Temporal composability	Not supported	Supported
Extensibility	Excellent in non time-critical applications	Only simple if extension planned for in original design
Membership service	Not provided	Provided
Clock synchronization	Not provided	Provided in microsec. range
Replica determinism	Not provided	Provided
Latency jitter	Variable, load dependent	Constant
Speed	Up to 1 Mbit/sec	Up to 2 Mbit/sec
Media access	Carrier sense multiple access with collision avoidance	Time division multiple access (TDMA)
Frame size	44 control bits plus 0 to 64 data bits	21 control bits plus 0 to 128 data bits
Frame types	Data Remote Error Overload	Initialization (I) Normal (N)
Bit encoding	NRZ with bit stuffing	Modified Frequency Modulation (MFM)
Error handling strategy	Immediate retry	Replicated channels, fail-silence
Error containment	No provisions	Containment of control errors across the CNI
Babbling idiot avoidance	No provisions	Independent bus guardian

Table 2: Comparison of CAN and TTP.

TTP: In TTP the data-sharing CNIs act as control-error containment boundaries. Since there is no control signal passed across the CNI, it is not possible for a control error to propagate from one node to another node or to interfere with the temporal control pattern of the communication system.

To contain data errors, a special High Error Detection Coverage Mode (HEDC) has been developed for TTP based systems. The HEDC mode provides two mechanisms to increase the error-detection coverage with respect to transient faults in the data field of a message:

(i) The calculation of an end-to-end CRC by the application task at the sending host to protect the complete path of the message between the sender task and the receiver task. The end-to-end CRC is calculated in addition to the 16 bit communication CRC of TTP. In a safety-critical application, the messages are thus protected by two CRC fields, one at the communication level, and one at the end-to-end (application) level. To ensure data integrity, the receiving node checks the end-to-end CRC at its CNI.

(ii) The time-redundant execution of application tasks at the sender. This service can be provided by the operating system in the host without any modification of the application software. If the end-to-end CRCs of the messages produced by the two task executions are not identical, then, one of the task executions has been corrupted by a transient fault. In this situation, it cannot be determined which one of the executions is incorrect. Therefore, both results are considered suspect, and none of the messages is sent. Since in a fault-tolerant configuration there is a replicated node providing the identical service, no service interruption is seen by the client.

Fault injection experiments have shown that both mechanisms are needed in high-dependability systems (Karlsson 95).

7. SUMMARY AND CONCLUSION

Table 2 compares the characteristic features of CAN in TTP in tabular form. From this table it can be concluded that the CAN protocol is most appropriate for soft real-time systems that require flexibility and do not have substantial timeliness and dependability requirements. If composability, hard real-time performance and dependability are more prominent issues than flexibility, then the TTP protocol seems to be the more suitable choice.

ACKNOWLEDGMENTS

This paper has been supported in part by the ESPRIT OMI project TTA, by the Brite Euram project X-by-Wire and by the ESPRIT LTR project DEVA.

REFERENCES

CAN (1990). Controller Area Network CAN, an In-Vehicle Serial Communication Protocol. *SAE Handbook 1992* SAE Press. pp. 20.341-20.355.

Karlsson, J., P. Folkesson, et al. (1995). Integration and Comparison of Three Physical Fault Injection Techniques. *Predictably Dependable Computing Systems* B. Randell, J. L. Laprie, H. Kopetz and B. Littlewood Eds. Heidelberg. Springer Verlag. pp. 309-327.

Kopetz, H. (1997). *Real-Time Systems, Design Principles for Distributed Embedded Applications; ISBN: 0-7923-9894-7.* Boston. Kluwer Academic Publishers.

Kopetz, H. and W. Ochsenreiter (1987). Clock Synchronisation in Distributed Real-Time Systems. *IEEE Trans. Computers*. Vol. **36**. pp. 933-940.

Kopetz, H. and J. Reisinger (1993). The Non-Blocking Write Protocol NBW: A Solution to a Real-Time Synchronisation Problem. *Proc. 14th Real-Time Systems Symposium*, Raleigh-Durham, North Carolina.

Liu, C. L. and J. W. Layland (1973). Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. of the ACM*. Vol. **20**. pp. 46-61.

Miesterfeld, F. and H. R. (1991). Survey of vehicle multiplexign encoding techniques. *Automotive Technology International '92'* M. Scarlett Ed. London. Sterling Publications International. pp. 253-265.

Poledna, S. (1995). *Fault-Tolerant Real-Time Systems, The Problem of Replica Determinism*. Hingham, Mass, USA. Kluwer Academic Publishers.

SAE (1995). Class C Application Requirements, Survey of Known Protocols J20056. *SAE Handbook* SAE Press, Warrendale, PA. pp. 23.437-23.461.

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

Tindell, K. (1995). Analysis of Hard Real-Time Communications. *Real-Time Systems*. Vol. **9**. pp. 147-171.