An Efficient TDMA Start-Up and Restart Synchronization Approach for Distributed Embedded Systems

Vilgot Claesson, *Member*, *IEEE*, Henrik Lönn, *Member*, *IEEE*, and Neeraj Suri, *Senior Member*, *IEEE*

Abstract—A desired attribute in safety-critical embedded real-time systems is a system time and event synchronization capability on which predictable communication can be established. Focusing on bus-based communication protocols, we present a novel, efficient, and low-cost start-up and restart synchronization approach for TDMA environments. This approach utilizes information about a node's message length that forms a unique sequence to achieve synchronization such that communication overhead can be avoided. We present a fault-tolerant initial synchronization protocol with a bounded start-up time. The protocol avoids start-up collisions by deterministically postponing retries after a collision. We also present a resynchronization strategy that incorporates recovering nodes into synchronization.

Index Terms—Data communications, access schemes, real-time and embedded systems, distributed applications.

1 Introduction

The pervasiveness of computer control is extending to safety-critical embedded systems in mass-market systems, e.g., X-by-wire control in cars, for the enhanced functionality and flexibility it offers. However, the high cost of implementation and adapting to existing underlying bus-based communication protocols, as well as the overhead costs of distributed protocols limit their effective usage. For cost-sensitive mass-market safety-critical systems (e.g., computer control), we develop low-cost and fault-tolerant synchronization strategies for Time Division Multiple Access (TDMA) bus environments. Specifically, highly efficient and fault-tolerant communication primitives for safety-critical systems with hard real-time requirements. This paper builds upon the basic ideology presented in our initial work [1]; we have since developed a novel algorithm that significantly reduces the maximum start-up time.

Virtually all existing TDMA synchronization techniques define and utilize explicit bits for node ID, which are used in the synchronization algorithm. In this paper, we remove the explicit bits used for node ID; instead, we base our approach on the existence of unique message length patterns to be used as node identifiers. The communication primitives are intended for communication protocols where the access method is TDMA. The primary primitives of interest address the start-up behavior of a protocol in a TDMA environment. We investigate how to effectively use the information about messages lengths (ML) as a form of message identifiers on which a synchronization approach can be built—we term this the ML-approach. A solution for avoiding start-up collisions is also presented using a method based on the same idea.

Broadcast media approaches are prolifically used in many systems on account of their simplicity and low implementation/operational costs. In computer control, many different protocols utilize a broadcast media, for example, CAN [2]. However, with a shared communication media, different access strategies exist. The most common strategies utilize contention resolution, more specifically, Carrier Sense Multiple Access (CSMA), where each node senses bus activities and may send when none is detected. The drawback occurs when two (or more) nodes send at the same time, the messages will collide. Therefore, it is often combined with Collision Detection where the colliding nodes withdraw if they sense a collision; this is called CSMA/CD and is used in, for example, Ethernet. Although used with great success in Ethernet, the CSMA/CD method is not appropriate for hard real-time systems due to the inherent lack of determinism. With CSMA/CD, it is not deterministically possible to avoid repeated collisions, which effectively constrains estimating worst-case communication times.

To avoid the limitations of CSMA/CD for real-time systems, collisions can be avoided using bit arbitration. With bit-arbitration, messages sent simultaneously are arbitrated using the bit sequence in the beginning of each message. This implies a priority order among messages. Nodes are not allowed to send messages with the same arbitration bit sequence. Using the priority order among messages, a worst-case communication time can be calculated for each message [3]. Using bit arbitration, the bus propagation time imposes a minimum length of the communication bit. Furthermore, bit pulses must be fairly well formed for the arbitration to work. These two facts limit the possible bit communication speed.

Token bus (IEEE 802.4) and mini-slotting [4] are other fundamentally different medium access schemes. However,

- *V. Claesson and N. Suri are with the Department of Computer Science, TU Darmstadt, 64283 Darmstadt, Germany. E-mail: {vilgot, suri}@informatik.tu-darmstadt.de.*
- *H. Lönn is with the Volvo Technology Corporation, Electronics and Software, 412 88 Göteborg, Sweden. E-mail: henrik.lonn@volvo.com.*

Manuscript received 11 Nov. 2002; revised 20 Sept. 2003; accepted 1 Dec. 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 117751.

token bus is sensitive to loss of the token and mini-slotting is limited in bandwidth as it is based on the concept of delays. In this paper, we focus on the TDMA media access method where nodes are preassigned time-slots in a repeating schedule. TDMA communication provides a deterministic behavior where, for example, arrival times and worst-case delays can be easily calculated. Although TDMA communication has been criticized for its static properties and its consequent lack of flexibility, the properties resulting from the determinism of TDMA communication are very useful, for example, evident timing, composability, easy fault detection, and testing [5].

Furthermore, most computer control systems have real-time requirements where these properties are particularly important. Combined with safety requirements in a hard real-time system, the consequences are catastrophic if deadlines are missed.

This paper is organized as follows: In Section 2, we review related work. This is followed by a presentation of the system and fault model in Section 3. In Section 4, the novel initialization and resynchronization approaches are described in detail. Sections 5 and 6 present the achieved bounds on the start-up and the related evaluations. Section 7 details the supporting simulation results.

2 Related Work

Several solutions to TDMA system start-up exist. Many protocols utilize a bus master with a special sync message that identifies the start of a communication cycle, e.g., [6]. Another example is Byteflight [7], that can be run as a TDMA protocol. However, we have chosen to avoid bus masters as it limits the reliability by introducing single points of failure. Consequently, we have directed our attention to distributed approaches.

One possible approach without bus masters is to use a known nondestructive bit-pattern, i.e., a jamming signal, that indicates the beginning of a TDMA cycle such that more than one node can transmit this sync pattern, see [8]. This naturally increases the overhead as the synchronization pulse must be sent at each and every TDMA-round. A similar approach is to use a unique signal level, e.g., a third signal level, other than 0 or 1. However, the extra hardware necessary would probably be more efficiently used to improve the bit encoding. Also, for both the jamming signal as well as the additional signal level a faulty node may repeatedly issue the resynchronization signal. Such a failure would be more severe and difficult to mask than other failure modes that result in invalid transmissions. Alternately, if the synchronization information was embedded in a regular message, a message (and a correct checksum) would have to be transmitted successfully in order to achieve synchronization. This is unlikely unless the node is functioning correctly.

Another approach lets one or more nodes try to initiate the communication by sending a message, the receivers of the message can use it for synchronization. There is a risk for collisions and even repeated collisions using this method. This can be handled by different ways for the node to make retries, for example, using a random back-off time as in Ethernet [9]. However, this implies a lack of an upper bound on the start-up time which is undesirable for real-time systems with safety critical implications. Thus, we focus on developing a start-up approach with a deterministic upper bound on start-up time.

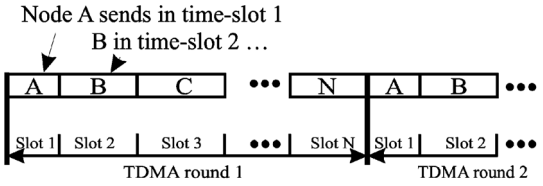


Fig. 1. A TDMA communication round.

In the TDMA protocol TTP [10], a node is reset and transits to a start-up mode on initialization or when a system-wide communication blackout has occurred. On entering this mode, each node has a unique delay until its first message is transmitted. The unique delay reduces the risk of collisions, but it also means that we cannot continue sending according to the original bus schedule. Instead, the bus clock must be reset when the initialization mode is entered. Moreover, if collisions are detected while in this mode, all nodes must reset their clocks [11].

The TTP protocol is one of the most established and robust time-triggered approaches. It is a full communication solution containing not only start-up and restart mechanisms, but also services as membership handling and changing operational modes of the communication. It should be clear that our approach only targets start-up and does not address the membership and operational modes. However, there is nothing preventing this synchronization approach to be combined with other communication services, and even TTP's approach for operational modes and membership handling.

In the next section, we present the pertinent system and communication model for our proposed approach. We also detail the fault types considered.

3 System, Communication, and Fault Models behind the Approach

The System Model: The target systems are safety-critical with real-time requirements. The system consists of n autonomous nodes that communicate via a broadcast bus. The nodes have local counters that are used to control the sending and receiving of messages. Furthermore, the communication system operates in a cyclic manner, where nodes have preassigned time slots in a TDMA communication round, see Fig. 1. Each node has a static list indicating when to send, and also when and from whom to expect messages in the cyclic TDMA round. If nothing else is stated, we will assume that each node i sends messages of different lengths as denoted by tm_i.

Communication Model: The synchronization of the nodes in the communication system can be divided into a number of steps, namely: 1) collection of nodes clock values, 2) calculating adjustment value, and 3) clock adjustment. Thus, clock synchronization requires that the receiver of a message knows the sender's identification (ID) and local time (e.g., when the message was sent). For this type of TDMA communication, this information can be extracted from the arrival time of messages as they are known a priori. The differences between the local and sender's clocks are calculated by the differences in the expected arrival time and the real arrival time.

We assume that synchronization is handled by a standard clock synchronization algorithm controlling the progress of the local clocks, for example, using the daisy-chain clock synchronization algorithm [12]. Other examples

of existing suitable synchronization algorithms can be found in [13], [14], [15].

Synchronization ensures that every node has the same view of the current position in the communication schedule, which suffices for nodes to know when to send their messages.

Each node must store the synchronization schedule containing the information of when and what a node is expected to send. The storage requirement for this information is normally relatively small, typically a few kilobits, since we deal with embedded systems with a limited number of nodes.

The global time-base can be used to synchronize distributed tasks and minimize delay and jitter by relating execution times of tasks to the TDMA rounds, as has been established by [10], [11]. This simplifies the scheduling of periodic tasks.

The intended area for these communication primitives is hard real-time systems that also need to be cost-efficient. The intent is to keep the primitives as simple as possible. Communication bandwidth for embedded systems is sparse and has developed relatively slowly compared to the growth in processing capacity. Thus, it is important to have efficient protocols with limited communication overhead.

The Fault Model: We assume that nodes follow *fail-silent* semantics that prevents faulty nodes to fail in a mode where they continuously transmit on the bus. Such a failure would overflow the bus and prevent any normal communication, including synchronization traffic. The fail-silent property relies on high coverage of the nodes' error detection mechanisms. It can be argued that, since sufficient coverage may be difficult to achieve, such failure semantics are unsuitable for safety-critical system. However, recent work indicates that using rigorous design and error detection methods, a very high coverage can be achieved for this fault model [16].

The communication media has omission failure semantics such that messages are either received correctly and on time or not at all. Byzantine failures are not considered as they are avoided by design, using a bus combined with message checksums and similar design means.

The initial synchronization algorithm requires a majority of nodes to synchronize such that different smaller groups do not form disparate cliques. This puts a limit on the number of tolerated faulty nodes to $\lfloor (n-1)/2 \rfloor$, where n is the number of nodes in the system. Thus, the initial synchronization will tolerate $\lfloor (n-1)/2 \rfloor$ node failures or message omissions. The number of message omissions may affect the synchronization time, which will be discussed in Section 6. A node recovering from a failure will need resynchronization in order to send messages. A node using our ML-approach for synchronization will regain synchronization after a unique pattern of message lengths has been received. Thus, assuming normal operation, message omissions will only affect the time for resynchronization of the node.

In the next section, we discuss our approach along with existing solutions for initial synchronization and resynchronization. We highlight properties and shortcomings, in order to put our approach in perspective.

4 Initialization and Resynchronizations

This section outlines the basic idea of our proposed synchronization approach; we precede that with a short discussion on existing synchronization solutions.

Current Approaches: Sending the time of local clocks explicitly in messages is an established technique to exchange time and position information in the communication schedule. A distributed algorithm can then be used to agree on the global time. We advocate an approach where the message arrivals are clocked and these time values are used for synchronization. As messages are prescheduled, these time values will reflect the local time of the corresponding sender. Consequently, we must be able to identify the sender of the message. Then, the static schedule unambiguously provides the send time. The difference from the scheduled send time and the local time when the message was received, is used to create a correction of the receivers' clock. The correction can then be achieved using an averaging algorithm; other possible approaches are described in, e.g., [14]. In this paper, we focus on how initial synchronization can be made more efficient by using our approach for transferring message **id**s. The synchronization properties such as precision, etc., will naturally be inherited from the chosen clock synchronization algorithm.

In most communications systems, a message identifier is included in the beginning of messages, as **id**-fields, e.g., [2], [4], [17], [18], and [7]. However, using statically scheduled messages, the reception time of the message can serve as the senders **id**, thus making the sender **id**-field of messages unnecessary during normal operation when the clocks are synchronized. Thus, before the nodes achieve synchronization, explicit sender **id**s are necessary.

To handle initial synchronization and resynchronizations, we can include a sender-**id** in all messages, as done in DACAPO [18]. Another way of achieving synchronization is to send special initial messages, as done in TTP [10]. Sending the **id** in all messages will add extra overhead that is useful only at start-up and at synchronization. Thus, sending special messages at start-up appears as a good idea, but then we increase the complexity by adding an extra communication mode at start-up. Furthermore, resynchronization of nodes is not supported if there is no sender **id** in messages, i.e., reintegration of a node that has lost synchronization. A third alternative is to let a node send periodic messages with resynchronization information. If the node that sends such a message fails, nodes that have lost synchronization will not be able to reintegrate. Therefore, additional nodes have to send messages with resynchronization information to tolerate failures. The disadvantage is the requirement of an extra synchronization mode and the additional overhead at runtime.

Proposed Approach: Our solution takes advantage of information that is inherent in statically scheduled communication such that inclusion of the sender's **id** in messages becomes unnecessary. This is achieved by using the message length as the **id** of corresponding senders. A receiver can then identify the sender of a message by the unique sequence of message lengths received from the communications media. Each node has a list that contains all nodes, their message lengths, and the expected receive order such that the sending node can be looked-up in this list. After a unique message length (or sequence of messages) has been received, the sender can immediately be identified using this table. The unique message lengths

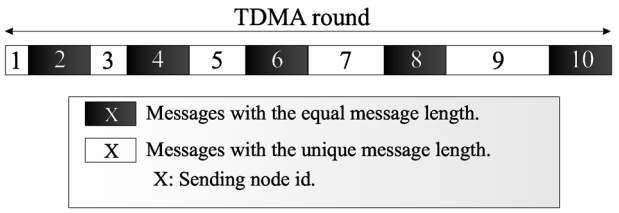


Fig. 2. A TDMA communication round. Nodes with odd **id** numbers have a unique message length, all nodes with even **id** numbers have equal message length.

of nodes' are chosen to reflect the individual nodes bandwidth requirement, i.e., the node with most data to transfer (per time unit) will have the largest message. The only case when it is impossible to create a unique sequence of messages is when all nodes require the same message length. However, in such a case, this method is still possible but the symmetry must be broken by, e.g., differentiating the message length of one or more nodes. See Section 4.1 for a detailed discussion of this issue.

Using this approach, nodes will know the position of the communication schedule as soon as one unique message length or sequence of messages has been received correctly. However, using different message lengths will not prevent messages from colliding. Making a new retry one TDMA cycle later can lead to a new collision and, in the worst-case, lead to an infinite sequence of collisions. A common workaround is to wait a random delay before resending, usually called exponential back-off.[1] This works to attain synchrony, though does not provide for the bounded start-up and resynchronization times needed in safety critical real-time systems.

To achieve a bounded start-up when messages collide, each node will delay its retry by a short but unique time period. We will show that, by using time periods that are short compared to the cycle time, a short and bounded worst-case start-up time is achieved while the average startup is fast. Although, this worst-case scenario is extremely unlikely, we support our proposition with extensive simulation results. We show the average start-up times of this approach using these simulations.

4.1 Identical Message Length Scenario

In most systems, nodes are likely to have different demands on bandwidth and the requirement of a unique message length sequence is normally not a limitation. However, there could naturally exist situations where a number of nodes in a system having similar assignments generate similar communication loads. One example could be brake nodes in a car, but in this case, other nodes in the car will likely break the symmetry. Properties contributing to nodes requiring the same message length are, for example, larger system size where the increasing number of nodes and messages makes it more likely that different nodes want the same message lengths. Furthermore, data transferred in most systems is often in multiples of bytes which also reduces the number of possible message lengths. As our approach works with unique sequences of message lengths, there are still cases where we actually need to separate messages. In Fig. 2, we show an example where nodes with even **id** numbers have equal message length. Nodes with odd **id** numbers demand different message sizes, therefore a unique message sequence is easily formed, for example, by sending a unique message every second message.

4.1.1 Handling of Identical Message Lengths

In an unlikely situation where all nodes require identical messages, we need to separate *at least* one message to form a unique sequence. A simple solution is to make one message slightly longer, i.e., add padded bits, such that a unique sequence is created. When creating unique sequences, we can trade the average time to detect a unique message against the additional overhead for padded bits by controlling the number of messages with unique message length.

When considering the initial start-up, nodes can still use messages with the same length by adding simple constraints on sending at the initial synchronization. Note that when one identifiable message (by length or sequence) has been received by a majority of the nodes, it is sufficient to achieve system level synchronization. Thus, we can constrain certain nodes during the initial start-up such that messages directly can be identified. For example, if a nonunique message is sent, all nodes (including the sender) will assume that it is the last message in the TDMA round of that length. Hence, all nodes can use that message for synchronization, although they cannot use that message payload. This simple method will allow many messages with the same message length.

We are targeting systems where data exchange often consists of control values, e.g., actuator set-points, etc., such data typically needs around 8 to 32 bits per value. These values are packed in messages and transferred on the communication media. Normally, a number of these data values are combined into a message, which determines the message lengths. Each data message could be sent separately, but that would obstruct our main objective to decrease the overhead, as each message imposes an overhead, e.g., for checksums, etc. Thus, better message response may come at the expense of decreasing bandwidth efficiency.

As larger systems are considered, which increases the number of messages, we may use messages with equal length. Thus, we use a unique sequence of messages to decide the system's current position of the communication schedule. When identifying the unique sequence, a penalty follows as more than one message might be necessary to decide the position in the schedule. In Section 4.1.2, we show how this time penalty can be reduced under certain conditions.

There are a few situations we need to avoid that form identical sequences that are repeated, which makes it impossible to distinguish the exact position of the schedule. Assume that a number of nodes send messages with length x, and another set of nodes sends messages with length y and, similarly, there are nodes using message length z. Thus, x, y, and z are nonunique message lengths, i.e., they do not provide unambiguous information of the position in the schedule. In Fig. 3, we show a few examples of unique and nonunique sequences. The difference among unique sequences is the worst-case time to unambiguously decide the position in the communication schedule.

Note that we must use at least one more node with nonequal message length than the number of faulty nodes that are to be tolerated.

1. The exponential back-off strategies used in CDMA do not provide a guaranteed bounded time to bus access. Predictable bounded times are essentially required for safety critical applications.

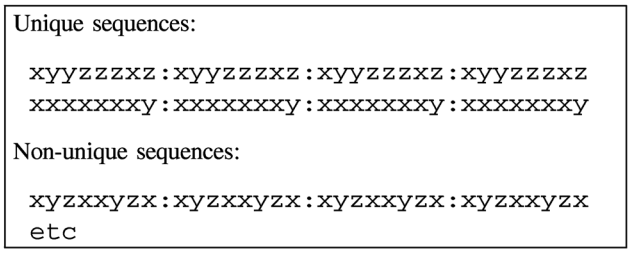


Fig. 3. Unique and nonunique sequences of message received from a bus. Each TDMA-cycle is eight messages and they are separated with semicolon (:).

4.1.2 Optimizations

In this section, we describe how to minimize delays emanating from using messages with the same length. This will ensure that the initial start-up is fast despite messages with equal length. A sender with a nonunique message length, say node b, will be preselected if a node receives a message with the length of node b, i.e., nodes will assume b was the sender independently of who actually sent it. This preselected node, b, will be determined before runtime. When receiving a nonunique message, all nodes will assume b as the sender and, thereby, the system nodes can be synchronized using any of the messages with that length. For example, if we have a 6-node system with nodes labeled, "a" to "f," i.e., the node id:s a, b, c, d, e, f, sending in alphabetical order. We assume nodes a and b have the same message lengths. Then, if the nodes receive a message from node a or b in the start-up phase, they will always assume that b was the sender, regardless of the actual sender. All nodes will synchronize to this message and they assume c is the next message to be received.

The drawback of this method is that the message content cannot be used when the first received message is not unique, as the sender is uncertain. This is not a problem during an initial start-up or system resynchronization as nodes are synchronized after the first message is received. Thus, the contents of following messages can be used. We argue that this is reasonable since the primary issue is to get nodes synchronized and it is only one message where the data contents cannot be used due to an unknown sender. Thus, the only negative effect of this in the initial start-up is that the first message cannot be utilized.

This optimized approach cannot be used for resynchronization of a single recovering node, as such a node must pinpoint the exact location in the schedule. Thus, a resynchronizing node must wait until a unique pattern/message has been received in order to be sure of the current position in the communication schedule.

4.2 The Node-Level Synchronization Operations

On this background, we now detail the node synchronization process. Nodes work in three different modes depending on the level of synchronization achieved, see Fig. 5. In Mode (1) *normal operation* a node is synchronized and sends according to the preassigned schedule. Mode (2) *resynchronization mode* is entered when a node has lost synchronization or messages from less than half of the nodes are received. Mode (3) is the *recovery mode* that a node enters at start-up and after a disturbance preventing message reception for the duration of a full TDMA cycle. The rules are based on those described in [19]. We first provide some relevant definitions that are used in the synchronization protocol description in Section 4.2.2.

4.2.1 Definitions

n: The number of nodes in the system.

T: The TDMA round time.

$\mathbf{tm_i}$: The time it takes for a node i to send its message M_i, i.e., this time is proportional to the message length.

$\mathbf{INC_i}$: Each node i is allotted a unique time period INC_i used to delay the nodes transmission after a collision.

$\mathbf{tm_{max}}$: The time it takes for the node with the longest message M_{max} to send its message, i.e., the time it takes to send the longest message.

$\mathbf{TC_i}$: The time counter for node i, keeps track of the current time in the schedule. With this pointer, each node will know when to send.

$\mathbf{ST_i}$: The send time of node i.

$\mathbf{t_c}$: A variable that stores the start time of the first detected collision event.

SP: The silence period (SP) is the time a node must check the media for existing traffic, before it is allowed to make the initial send. $SP = T + INC_{max}$.

MRvec: Message Receive Vector, containing information on whether the latest n messages were received correctly or not. In a (n) node system, a node's $MRvec$ contains n zeros and ones, e.g., $\{0, 1, \ldots 0, 1, 1, 1\}$, where 1 represent correctly received messages and a 0 incorrectly or missing messages. $MRvec$ is a FIFO list where a new 0 or 1 is shifted ($\overrightarrow{shift}$) into the vector representing the receive status of the latest received or expected message. We denote the number of 1 in $MRvec$ with $ones(MRvec)$.

$\mathbf{SC_i}$: The silence counter for node i, timing the period from last bus event or disturbance. The silence counter is reset every time any messages/traffic is sensed on the bus.

Having introduced the definitions, we now present the synchronization protocol, with the operational modes and their corresponding operation.

4.2.2 Synchronization Protocol, Modes, and Operation

Each of the following operations is performed at each node. A node will also treat message reception from itself in the same way as other nodes, e.g., updating the $MRvec$. This protocol is triggered by events set according to the cyclic clock (timeouts) and media activities (message received).

Recovery mode: (Fig. 4) The start mode which nodes enter initially at start-up. A node starts by listening for existing traffic during slightly more than one communication cycle. If no message was received during that time, the node will send its message according to its own clock. A node returns to this mode in case of complete loss of synchronization.

Resynchronization mode: Having sent a message in recovery mode, a node enters resynchronization mode and waits for reception of messages from half of the nodes. (Note: No message transmissions are made in Resynchronization mode).

Events handled by node i in **Resynchronization mode**:

Message Received: message sender is node s

begin
$TC_i := ST_s + tm_s$
$1 \overrightarrow{shift}\ MR_{vec}$
if $ones(MR_{vec}) > \lfloor n/2 \rfloor$ **then**
⇒ "*Normal Mode*"
fi
end

Timeout waiting for message: expected message from node e according to local clock TC_i, but received nothing
or an incorrect message.

begin
$TC_i := ST_e + tm_e$
$0 \overrightarrow{shift}\ MR_{vec}$
end

Timeout silent media:

begin
if $SC > ST_i$ **then**
⇒ "*Recovery Mode*"
else
$ST_i := t_c + T + INC_i$
fi
end

Normal mode: When nodes are synchronized and messages are received from more than half of the system nodes.

To reach normal mode (see Fig. 5), we require message receipt from a majority of the nodes, thus we tolerate $\lfloor (n-1)/2 \rfloor$ faulty nodes. The effect of failure during start-up and resynchronization is a delay in the start-up time.

It is important to avoid repeated collisions of messages which would prevent the system from synchronizing. This is handled in the recovery mode above. Each node i is assigned a time increment INC_i, significantly shorter than the period time T. If a collision occurs at time t_c, colliding nodes will postpone their next retry with a time equal to the increment INC_i. Thus, the send time of the retry, t_r, for node i will be one period plus the time increment, i.e., $t_r = t_c + T + INC_i$. This will postpone the resending of the message with the time period INC_i. Nodes that are not involved in the collision will mark the collision time t_c and refrain from any transmissions during the time they can expect an retry from the colliding nodes. As colliding nodes delay their next send by a node unique value, INC_i, they will not collide again. Thus, after a deterministic time interval, as detailed in Section 5, (2), at least one node will access the media and transfer its message. All nodes will receive this message and synchronize to it preventing further collisions. Thereby, the initial synchronization process completes.

Events handled by node i in **Normal mode**:

Message Received: message sender is node s

begin
$TC_i := TC_s + tm_s$
$1 \overrightarrow{shift}\ MR_{vec}$

end

Timeout waiting for message: expected message from node e according to local clock TC_i, but received nothing
or an incorrect message.

begin
$TC_i := ST_e + tm_e$
$0 \overrightarrow{shift}\ MR_{vec}$
if $ones(MR_{vec}) \leq \lfloor n/2 \rfloor$ **then**
⇒ "*Resynchronization Mode*"
fi
end

Send Timeout: The local time TC_i is equal to the ST_i

begin
SEND
$ST_i := ST_i + T$
end

In Section 5, we now derive the maximum number of collisions in a worst-case scenario and substantiate its correctness.

5 Upper Bound on Start-Up

In this section, we establish the upper bound on subsequent collisions. In an initial start-up scenario, a node will start in the Recovery mode. In this mode, a node will synchronize with the first received message. Thus, we need to show that the nodes, in a bounded time, will receive an uncorrupted message to synchronize with.

To prove this upper bound on the initialization, we use the following assumption:

$$INC_1 < INC_2 < \cdots < INC_n << T, \tag{1}$$

where n is the index of the last node. Thus, node n will use the longest delay after a collision. For clarity, we have used the specific terms INC_{min} and INC_{max} instead of INC_1 and INC_n, respectively. The minimum time unit must be at least one propagation delay, τ, of the media. Thus, the shortest INC_{min} must be at least τ, and the following, INC_{min+1}, must differ with at least τ. Furthermore, clock drift will effect the start-up. Thus, the difference d between different increments, i.e., $d = INC_{i+1} - INC_i$, must be large enough to avoid the effects of clock drift. We need to avoid that the clock drift can make the nodes collide again, thus $d > T \cdot 2\rho + \tau$, where T is the TDMA length and ρ is the maximum clock drift.

When nodes p and q collide, we want to prevent further collisions from occurring. This is achieved when the colliding nodes, p and q, postpone their send retry with INC_p and INC_q, respectively, after a collision. As $INC_p \neq INC_q$, nodes p and q are prevented from colliding subsequently.

Recovery mode:

Message Received: message sender is node s

begin
$TC_i := ST_s + tm_s$
$1 \overrightarrow{shift} MR_{vec}$
if $ones(MR_{vec}) > \lfloor n/2 \rfloor$ **then**
⇒ "*Normal Mode*"
fi
end

Timeout waiting for message: expected message from node e according to local clock TC_i, but received nothing or an incorrect message.
begin
$TC_i := TC_i + tm_e$
$0 \overrightarrow{shift} MR_{vec}$
end

Collision detected on media:
begin
if node i not sending **then**
$t_c := t$
else
$ST_i := t_c + T + INC_i$
fi
end

Send timeout: time to send according to the local clock TC_i.
Note: there must have been a silence period of SP for the nodes to send.
begin
if $SC_i > SP$ **then**
if $t_c \neq 0$ **then** *% Collision detected?*
if $(t_i > t_c + T + INC_{min})$ & $(t_i < t_c + T + INC_{max})$ **then**
$ST_i := ST_i + T$ *% Send prohibited. Set new send time.*
else
$SEND$
$ST_i := ST_i + T$
⇒ "*Resynchronization Mode*"
fi
else *% Collision not detected*
SEND
$ST_i := ST_i + T$
⇒ "*Resynchronization Mode*"
fi
else *% Silent time not expired*
$ST_i := ST_i + T$
fi

end

Fig. 4. Events handled by node i in **Recovery mode**.

We summarize important properties of the protocol:

1. There will be one or more nodes that will be the first to send after a start-up.
2. If a single node starts to send, other nodes will listen and synchronize accordingly.
3. If two or more nodes send at the same time, there will be a collision at time t_c.
   - Nodes that participate in the collision will adapt to the collision and postpone their retries with the additional INC parameter.
   - For all other nodes, they will remember the collision time t_c and avoid to send during nodes retry, i.e., $[t_c + T + INC_{min}, t_c + T + INC_{max}]$.

Now, the colliding nodes, i.e., nodes p and q, will then send at t_p and t_q, respectively. Where:

$$t_p = t_c + T + INC_p$$
$$t_q = t_c + T + INC_q.$$

As we previously stated, $INC_p \neq INC_q$ and the nodes will not make their retransmission at the same time. Thus, all nodes will synchronize to the first of p and q that sends.

Now, if there was another node r that wants to send at t_p (or t_q), this would create another collision. However, this node started its media monitoring at the time t_r which is:

$$t_r = t_p - SP.$$

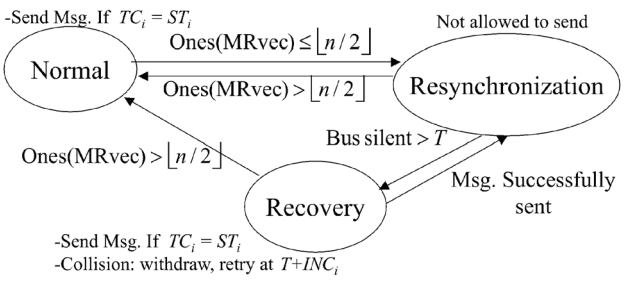


Fig. 5. Communication controller states.

However,

$$t_r = t_p - SP = t_c + T + INC_p - (T + INC_{max}).$$

Finally, we have:

$$t_r = t_c + INC_p - INC_{max} \leq t_c.$$

This means that the monitoring of the media, by node r, started before the transmission/collision occurred. This leads to a contradiction as, according to the protocol, it could not happen. Thus, none of the other nodes can create further collisions.

Nodes will fail by becoming silent and will therefore not affect the start-up scenario.

The worst-case time is then the longest t_p, i.e., with INC_{max}. Furthermore, we need to find out the maximum possible time until the first collision, i.e., t_c, may occur.

- First, we have the $SP = T + INC_{max}$.
- After SP, a node local clock may be in any position, thus we might need to wait an additional $T - ml_{min}$.
- Then, the first message is sent, which requires at most ml_{max} time units.

Thus, the maximum time from the point where at least one node starts executing, would be:

$$\begin{aligned} t_c &= T + INC_{max} + T - ml_{min} + ml_{max} \\ &= 2T + INC_{max} + ml_{max} - ml_{min}. \end{aligned}$$

Thus, the maximum start-up time ts_{max} is:

$$\begin{aligned} ts_{max} = \\ 2T + INC_{max} - ml_{min} + T + INC_{max} + ml_{max}. \end{aligned}$$

After collecting terms, we get:

$$ts_{max} = 3T + 2INC_{max} + ml_{max} - ml_{min}. \quad (2)$$

6 Properties and Overhead

In this section, we present the properties of this synchronization approach. The overhead of synchronization can be manifested in three ways, namely, 1) communication overhead, 2) synchronization time overhead, and 3) computation/memory overhead. The communication overhead relates to additional data that must be transferred in order to achieve synchronization. The time overhead relates to the additional timeouts, retransmissions, etc. Finally, the computation/memory overhead is related to processing and memory storage which is required in order to accomplish resynchronization.

This synchronization approach is efficient in the sense that it has low communication overhead, e.g., no id bits are necessary. However, there is invariably some overhead related to communication services like synchronization. In safety-critical real-time systems, the synchronization overhead and communication overhead are normally most critical. Computation time and memory is relatively cheap in comparison. Therefore, we favor a solution that may have a larger computation overhead but smaller time and communication overhead. The following list summarizes the main overhead contributors in this approach.

Communication: No additional messages are sent and no extra information bits are needed to achieve synchronization, except in the rare cases when we need to add extra bits to get unique message lengths.

Time: Synchronization is accomplished as soon as a message is received. It may be delayed by collisions, but is bounded as given in (2), Section 5.

Computation/Memory:

- Each node carries a list of "n" entries, i.e., one for each node in the system entries containing all message lengths and the corresponding nodes, it also includes the knowledge about the time slot it should be sent out in.
- Counters which keeps track of local time (TC) (i.e., point in the communication schedule) and silence on the media (SC).
- A storage value for the start-up increments, INC_i, the time for a collision t_c and the silence period SP.
- A vector with the n last received/expected messages, with only one bit necessary for each node, indicating received or not received messages.

To analyze the proposed Message Length (ML) approach, we compare it with two existing approaches for initial synchronization and resynchronization. The approach used in DACAPO [18] (*D-appr.*) uses fixed length messages where one or two special nodes change its send time in order to resolve collisions. The second approach is based on the start-up approach used in TTP [10], [20], the (*T-appr.*), see also Section 4.1 where nodes send special start frames and each node uses dedicated back-off times. Finally, an approach that behaves as ML except that it uses exponential back-off to resolve collisions. This approach will be referred to as ML_{exp}. The exponential back-off uses a random delay before retry, and additional collisions exponentially increase the time range from which the random delay is chosen. The exponential back-off is used in, for example, the Ethernet protocol [9]. This method would require a random number generator in each node instead of our increment value calculated and stored preruntime. Although slightly more complex in implementation, ML_{exp} is independent of system size.

In Table 1, the main differences in overhead for these start-up scenarios are summarized. The main categories we have compared are

1. communication bandwidth overhead (in number of bits),
2. storage overhead,
3. bounded/unbounded start-up time, and
4. maximum time for a node to resynchronize.

TABLE 1
Properties of Start-Up/Resynchronization Approaches

Method	Comm.(bits)	Memory (bits)	Start-Up	Resynch. (secs)
ML	0	$\propto n$	B	$T - tm_u$ to $\left(1 + \left\lfloor \frac{f}{2} \right\rfloor\right)$ T
D-Appr.	$log_2(n) \cdot n$	$log_2(n)$	B	$(f+1) \cdot tm$
T-Appr.	$log_2(n) \cdot (f+1)$	$log_2(n)$	B	$f \cdot t_d + tm_{im}$
ML_{exp}	0	$\propto n$	U	$(f+1) \cdot tm$

(B = Bounded and U = Unbounded.)

In our ML approach and ML_{exp}, there are no communication overheads since information is transferred using message lengths. The required storage is proportional to the number of nodes n such that a message can be identified by the message length. The major difference between these two is that ML has a bounded start-up time, see Section 5.

For the resynchronization, the maximum time it takes to resynchronize a node is important. Using the ML approach, the maximum resynchronization time is determined by the maximum time to detect any unique sequence of messages. Thus, it is dependent on how message lengths are set and distributed in the send schedule during design time.

We first look at a case where all nodes have unique message lengths. The time to send messages from node i, i.e., the duration during which that message occupies the bus, is tm_i and we assume that we index the nodes such that the send duration for node i is shorter than for node $i+1$, i.e., $tm_1 < tm_2 < \cdots < tm_i < tm_{i+1} < \cdots < tm_n$. Thus, the worst-case resynchronization time t_r, i.e., the longest time it can take for a node to receive a message, would be:

$$t_r = tm_{n-1} + tm_n. \tag{3}$$

In case of f faults, i.e., message omissions or node crashes, we have:

$$t_r = tm_{n-1-f} + \cdots + tm_{n-1} + tm_n. \tag{4}$$

As we see in (4), t_r will increase for each extra fault. As a pessimistic approximation, we can write $(f+1) \cdot tm_{max}$ if f is the number of tolerated faults and tm_{max} is the longest message length.

In the worst-case, there will be only one unique sequence thus, (i.e., one unique message); the worst-case resynchronization time is then one TDMA round minus the length of the unique message tm_u, i.e., $t_r = T - tm_u$. Now, we include faulty nodes leading to message omissions. On receipt of the unique message, a node can synchronize; similarly, on receipt of all the other messages without any omission, we know that the next message is the unique one. Thus, the worst-case synchronization time with one omission fault is: $t_r = T - tm_u + tm_u = T$. For two omission faults: $t_r = 2T - 2tm_u$.

$2tm_u$ reflects that we need an additional unique message for each tolerated faulty node. We can get a simpler expression by the fact that $t_r = 2T - 2tm_u \leq 2T$. Thus, for $f = \{1, 2, 3, 4, 5, \ldots\}$ we get $t_r = \{1T, 1T, 2T, 2T, 3T, \ldots\}$, etc. Then, we can bound the resynchronization time as:

$$t_r \leq \left(1 + \left\lfloor \frac{f}{2} \right\rfloor\right) T. \tag{5}$$

The TTP approach requires special initialization messages to be sent that include information about the sender. A node is synchronized when it has received an initialization message; this is equally true for a resynchronizing node. Thus, initialization messages must be sent during normal operation to allow a recovering node to resynchronize. Furthermore, to tolerate failure of the initialization message sender, $(f+1)$ nodes must send this type of message. The number of bits required in initialization messages is $(f+1)log_2(n)$, where $log_2(n)$ bits are needed to identify the sender and such a message must be sent $(f+1)$ times to tolerate f failures.

In the TTP approach, nodes synchronize to the first initialization message. In case of colliding nodes, TTP ensures that further collisions are avoided by ensuring that nodes listen to the bus and reset their local clock at a collision or bus event. Thus, after such a reset, nodes will wait for a node-specific time, based on the requirement, to send their message. For this to work and to avoid additional collisions, all nodes must have sensed the collision. By increasing the time between two successive initialization messages, we reduce the overhead stemming from initialization messages. However, this will increase the worst-case time for resynchronization of a recovering node. The worst-case resynchronization time depends on the time between and on the length of initialization messages, $f \cdot t_d + tm_{im}$, where t_d is the largest time between initialization messages and tm_{im} is the length of those messages.

In the D-approach, the *sender id* is always included in the message and will result in a communication overhead of $nlog_2(n)$ per TDMA cycle. There is no extra storage needed for either the *D-approach* or *T-approach* for the start-up. This should not be confused with the fact that TTP-controllers already store total information about the communication schedule, including messages lengths. This extra overhead is used for other purposes than the start-up synchronization and resynchronization.

Table 1 shows that the ML-approach combines a bounded start-up time with low communication overhead. For the resynchronization we should remember that the worst case for the ML-approach is based on a case where most nodes use the same message length. In a more likely case, e.g., with up to three nodes with the same length in a row, the worst-case resynchronization time should be three received messages.

7 Simulations

To show the normal start-up behavior of our approach, we have simulated the initial start-up synchronization and measured the time for all nodes to reach the Normal mode.

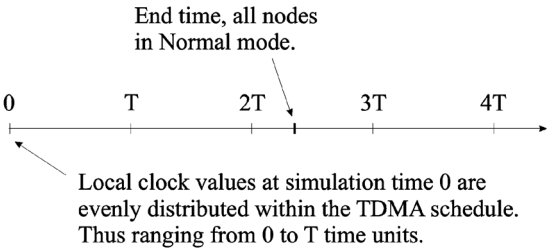


Fig. 6. Start-up initialization.

Note that the worst-case start-up time previously calculated was for the first message to be received by all nodes. The simulations have been done in a number of scenarios, such as under normal fault free condition as well as with faulty nodes during the start-up.

We have assumed a bus system where the bus is no longer than 40 m. The propagation delay for a 40 m cable is approximately 0.2 μs, and we selected a communication bit length of twice the propagation delay. This affects the start-up times, but as we have used the same for all simulated protocols, it does not affect the relative comparison between protocols. In these simulations, the basic time-units are 0.4 μs, i.e., the basic bit transmission time.

The message lengths are unique and chosen as a multiple of the basic time unit, of 0.4 μs. For the time increments (INC_i), it is important that they are longer than the propagation time. Furthermore, they should differ by more than one propagation time-unit each, such that after a collision between two nodes these nodes will not collide again. In our simulations, INC_i are chosen starting with one time unit, i.e., two times the propagation time, and for each additional node we add two time-units, e.g., $INC_i = 1, 3, \ldots, (1 + 2 \cdot n)$ for $i = 1$ to n where n, is the number of nodes.

The start times of the local clocks, i.e., the time counter TC, have been evenly distributed in the time interval (0,T) where T is the TDMA round time, see Fig. 6.

7.1 Normal Operation

We conducted our simulations by determining the start-up times under conditions when all nodes work correctly. To show how the ML-approach scales with system size we show the simulation results from a 6-node and 24-node system.

We assume B_{CT} is the number of bits that fits within a cycle, i.e., a Cycle Time (CT). Then, if b_t is the send time of a single bit, we get the number of bits during a Cycle Time as:

$$B_{CT} = \frac{CT}{b_t}.$$

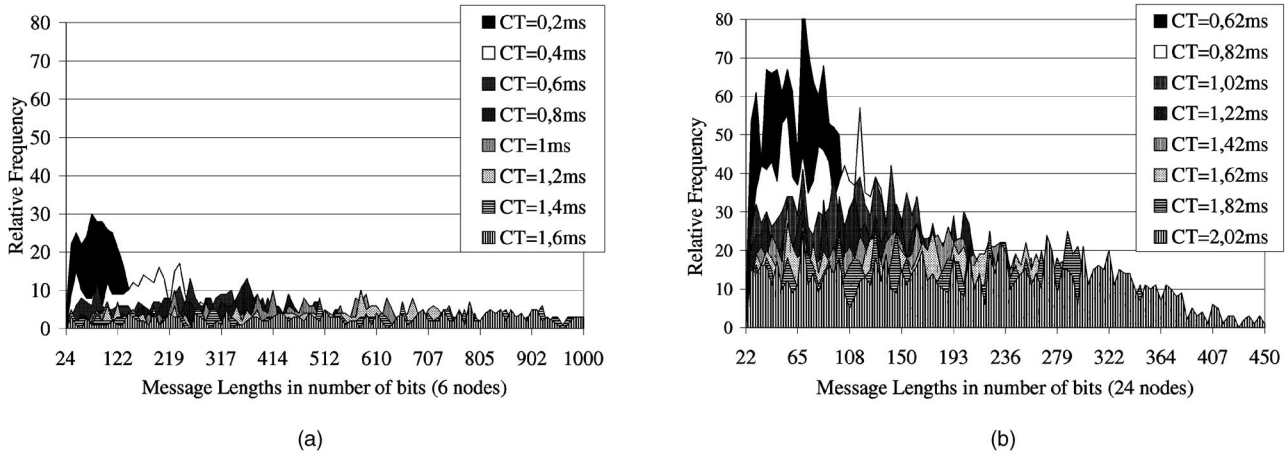
The message lengths have been chosen randomly from approximately 24 bits and up, such that the Cycle Time of a communication round is equal to a certain chosen period. The selected periods are between 0.2 and 1.6 ms. We have also varied the system sizes, using 6 and 24 node systems.

For each CT (cycle time), 50 sets of messages with randomly generated lengths were used. We have initially chosen to randomly generate messages between 24 bits and $\frac{B_{CT}}{(n/2)}$, where n is the number of nodes. The 24 bits is not a fixed limit, but has been chosen as it is reasonable to assume that smaller messages are rarely used. However, if messages of equal length were generated, they were separated by decreasing the length of one of them. Therefore, a few messages may have been separated such that their length are below the 24 bits. Normally, they are separated by increasing the length of one of the messages, so as not to interfere with the payload. However, to increase or decrease does not alter the simulations and the choice is based on the implementation.

The sum of all n messages S_r should be B_{CT}. Therefore, each randomly generated message length mlr_i was adapted according to (6). The used message length ml_i is then:

$$ml_i = (mlr_i - 24) \cdot \frac{S_r - 24 \cdot n}{B_{CT} - 24 \cdot n} + 24. \quad (6)$$

This ensures that messages are randomly generated starting from 24 bits and the total sum is B_{CT}. The message length distributions for the 6-node and 24-node system are shown in Fig. 7. We can see that with a shorter CT we have shorter messages in average. Furthermore, in the 6-node case, there is a wider range of message lengths 24-1,000 bits compared to 22-450 bits in the 24-node case.



(a) (b)

Fig. 7. Message length distribution with the ML approach. (a) A 6-node system and (b) a 24-node systems. Note the different scale on the abscissas.

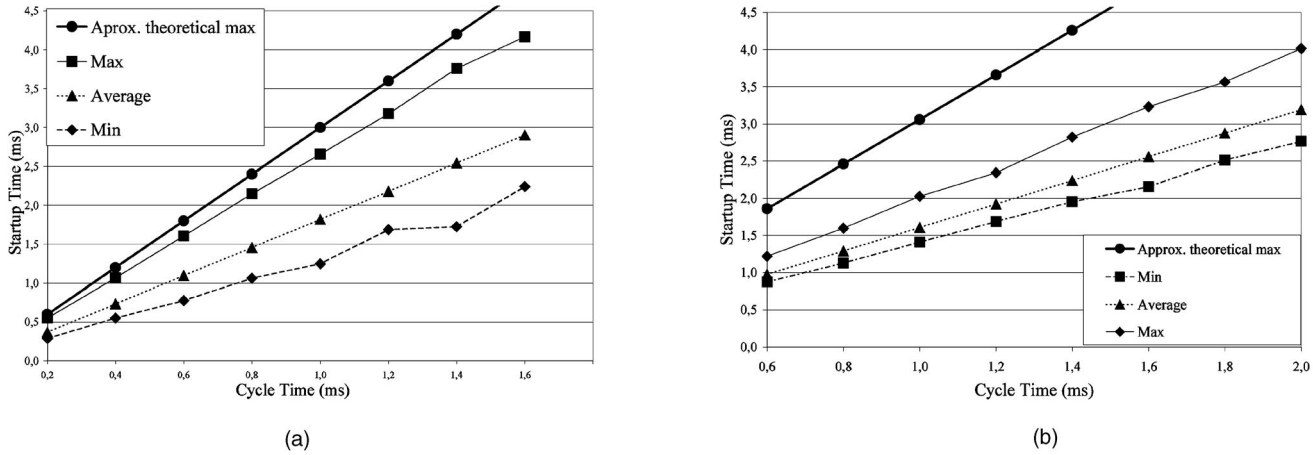


Fig. 8. Approximate theoretical max (for first message), Max, Average, and Min start-up time. (a) A 6-node system and (b) a 24-node system.

We use the 24-node case to show the operational capability of our approach for a relatively large system size as well. The system size of 6 and 24 nodes will also be used for the simulation of the fault scenarios.

For each of these 50 message sets 1,000 starts were run, such that in total 50,000 starts were run for each CT-case. In Fig. 8, we see the maximum, average, and minimum start-up times for these simulation runs. The average time to reach normal mode is almost linear to the CT. This means that the start-up time increases when the average message length increases while using the same system size. This also applies to the maximum and minimum start-up times.

We can calculate the maximum start-up time using (2). If we, for example, look at the 6-node case with a CT of 0.2 ms, from our simulation data, we see that the maximum time until the system reached the normal mode was 0.55 ms. The first node to send was node 3 followed by 4, 5, and 6. Thus, when the node 6 message has been delivered, all nodes will enter normal mode. To calculate the maximum start-up ts_{max}, we use $INC_{max} = INC_6 = 13 \cdot 0,4\ \mu s$, minimum message length is in this case 48 bits and the maximum message length is 130 bits. Thus, $ml_{min} = 48 \cdot 0,4\ \mu s$ and $ml_{max} = 130 \cdot 0,4\ \mu s$. and we get $ts_{max} = 0.64\ ms$.

Now, if we reduce the simulation start-up maximum from the impact of nodes 4, 5, and 6, we get the corresponding value, i.e., the time until the first message was received. Their total length is 288 bits which takes 0.11 ms and the worst measured start-up time is $0.55 - 0.11 = 0.44\ ms$. Thus, our measured start-up time of 0.44 ms is well below the theoretical maximum of 0.64 ms that is given by (2). What we show in Fig. 8 is a lower approximation of the theoretical start-up time of 3T ($\leq ts_{max}$). Thus, even when we use the shorter theoretical start-up time to compare with the simulation results (the time to enter normal mode), the simulation results stays well below the theoretical value.

As we can see in Fig. 8, the average start-up time is close to linear to the CT, i.e., the start-up time is a fixed factor of the CT. In the 6-node case, the start-up time is close to 1.8 times the CT and slightly less for the 24-node case where it is 1.6 times the CT.

Furthermore, we see that the theoretical maximum is above the corresponding simulated maximum values. In the system with few nodes, there is a smaller margin to the theoretical max value.

In Fig. 9, we show the relative frequency of the start-up times. As observable, the distribution of the different starts depends on the CT-length. The fact that start-up times are more spread in the 6-node case (except for CT=0.2ms) can

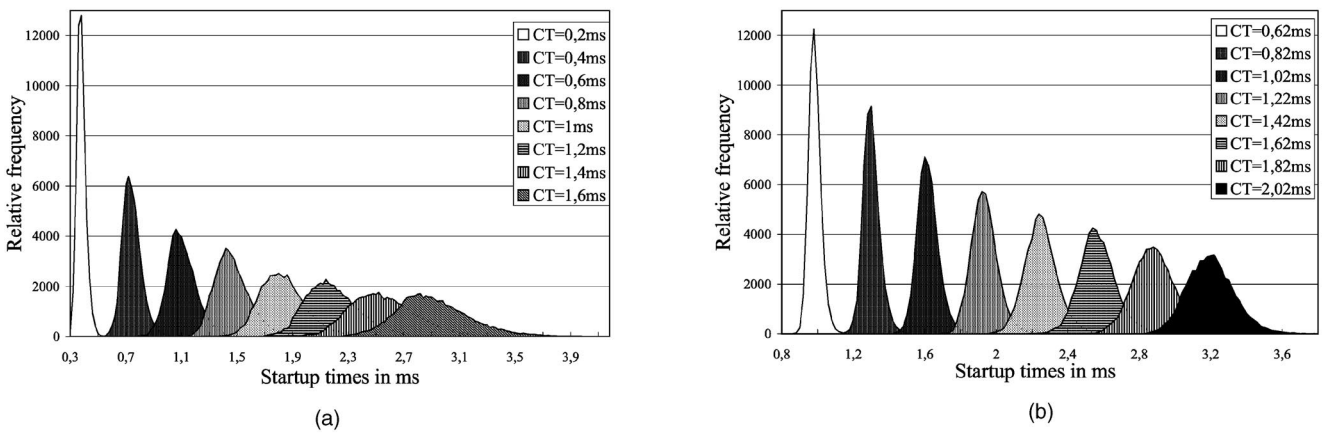


Fig. 9. The relative frequency of the start-up times in a (a) 6-node and (b) 24-node (b) system. In each graph, each curve corresponds to a Cycle Time, where the leftmost curve has the shortest Cycle Time.

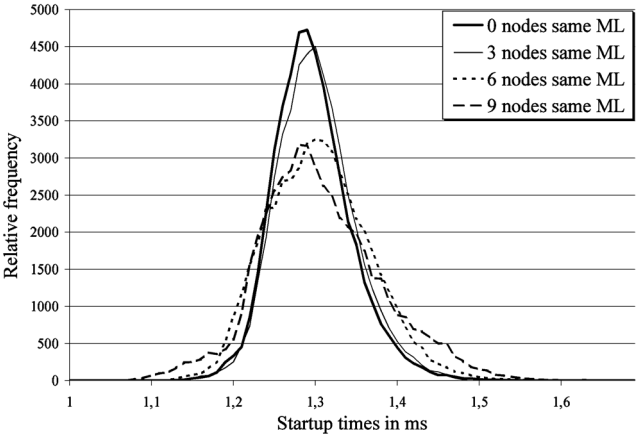


Fig. 10. The relative frequency of the start-up times in a 24-node system using three, six, and nine messages with the same length, respectively. The Cycle Time is 0,82 ms.

TABLE 2
Distribution of Nonunique Sequences Divided by the Different Systems Sizes

System size	Longest non-unique sequence, in # of messages.						
	1	2	3	4	5	6	7
10 nodes	55532	41742	2585	141	0	0	0
15 nodes	5233	82935	11177	624	30	1	0
20 nodes	21	74573	23872	1463	66	4	1
25 nodes	0	58843	38501	2511	132	13	0
30 nodes	0	42861	52780	4127	223	9	0
35 nodes	0	28957	64719	5982	330	10	2
40 nodes	0	18239	73271	7963	508	19	0
45 nodes	0	10669	78410	10320	573	27	1

The sequences are measured in the number of nodes required before they can be resolved. For each system size, 100,000 schedules have been measured.

be explained by the fact that there is a larger spectrum in message lengths for the 6-node case, see Fig. 7.

7.2 Equal Message Length Scenarios

We now outline the simulations conducted to show the effect of equal message length in a system. We have utilized the optimizations described in Section 4.1.2. Using this method will result in a limited effect on the average start-up times. To show this, we ran simulations with a 24-node system and a relatively small cycle time, see Fig. 10. This has been chosen as messages of the same size are more likely needed when using a system with many nodes that have requirements of relatively short messages, such that varying the message lengths is hard. All message lengths have been chosen randomly as described in Section 7.1; one message length is then used for a number of nodes. In our simulations, we have used the 24-node system with three, six, and nine messages with equal length. The simulation results are shown in Fig. 10. As the figure shows, there is a limited effect of using the same message lengths when we use this optimized approach. The main difference is when the number of messages with equal lengths increases, there is a slightly larger dispersal between the start-up times. For comparison, we have also included the case with no identical messages.

We can see that we get a small impact on the start-up times using the optimized method. However, there is still a question of the applicability of this method in reality. How often do we get schedules where we cannot use our method at all, i.e., without using some manual intervention like adding bits to get unique messages? It is hard to find typical real-time application data, for example, industry can seldom provide typical real-time system schedules or information about the amount of data they send. In order to get an indication of how often schedules occur that prohibit our approach, we have randomly generated a large number of schedules. We have then studied this data to see how often our approach can be used and how the maximum delays are imposed due to recurring message sequences.

To handle data values that are multiples of bytes long, many communication protocols use messages lengths with multiples of bytes as well. We use the same principle and use only messages with multiples of bytes. This naturally reduces the number of possible message lengths in the system. Messages in real-time systems are approximately 24 to 240 bits long excluding overhead, this gives us 27 different message lengths. However, to stay on the pessimistic side with our figures, we will restrict the message lengths further and only allow messages between 56 and 200 bits, which gives us only 18 different message lengths to choose from.

We have generated TDMA schedules, consisting of randomly generated messages that are evenly distributed among these 18 message lengths. These simulations have been done for different system sizes, i.e., with 10, 15, 20, 25, 30, 35, and 40 messages per TDMA-round. We have measured the longest nonunique sequences of messages in each simulation and studied the effect of increasing number of messages. The result is shown in Table 2, where we generated 100,000 schedules per system size. The columns show the occurrences of the longest sequence of messages that must be received before the sender's ID can be established. For each system size, the occurrences of all the generated schedules is shown such that the sum in each row is 100,000. In column 1, we see the occurrences where only one message must be received before the senders id can be established. In column 2, the longest sequence that exists in a schedule before the sender's ID can be established is 2, etc. For each system size we show in the table the distribution of the longest nonunique sequence. We found that even in quite difficult circumstances, all generated schedules could use our approach. However, when the system size increases, repeated sequences of message lengths becomes more frequent as possible message lengths are the same.

7.3 Fault Scenarios

In this section, we show a system start-up behaves when one or more nodes fail during start-up. A number of failure scenarios can occur during the start-up of the system.

The main failure scenario we have to consider is when a node falls silent before or during the start-up. Since this will affect the start-up behavior, we have simulated when faulty nodes are silent during the start-up. The main effect on the start-up is that the average start-up times for still working

TABLE 3
Simulation Result Using a 6-Node System with Zero, One, and Two Nodes Faulty during the Startup, the Cycle Time is 0.4 MS

Startup time (ms)	No. of faulty nodes		
	0	1	2
Mean	0,73	0,78	0,86
Std. Dev.	0,067	0,076	0,083
Min	0,40	0,58	0,57
Max	1,07	1,18	1,21

nodes increase, as can be seen in Table 3. It also shows the relatively small standard deviation on the average start-up for the three cases, i.e., zero, one, and two faulty nodes, and the increasing minimum and maximum start-up times, due to faulty nodes, during the simulations. To further show how the start-up is affected, we show in Fig. 11 the relative frequency of the starts simulated.

In order to show how this behavior scales to larger system sizes, we show in Fig. 12 the relative frequency of the startup times for a 24-node system. The maximum startup time increases with the number of faulty nodes and in our simulations the maximum startup time increased from 1.60 ms, for the case with zero faulty nodes, to 1.87 ms for the case with seven faulty nodes.

7.4 Comparison

We compare the simulations of our startup approach with the popular TDMA communication approach TTP/C [10], [20], [21]. We also compare with one of our earlier approaches, used in the DACAPO [18] system.

TTP/C is a Time Triggered protocol for distributed real-time systems. It focuses on safety-critical systems and is designed to tolerate faults. The information regarding the system startup behavior in TTP/C has been taken from [21] and [20].

To get nodes synchronized, TTP/C sends special messages with information about the time and the other C-state information. These messages are called *Initialization frames* (I-frames) and are sent at startup and regularly under

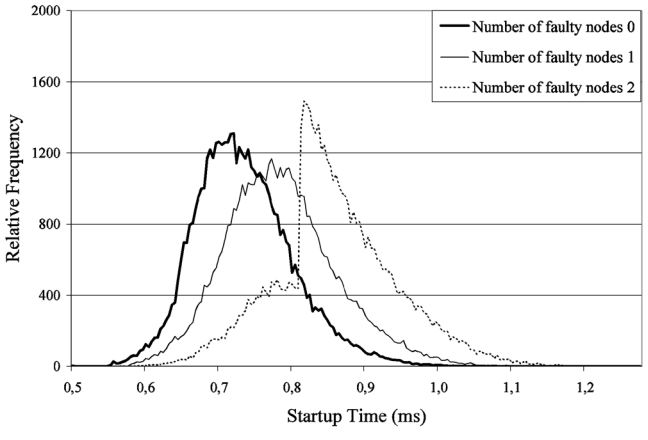


Fig. 11. The relative frequency of the startup times in a 6-node system. The cycle time is 0.4. ms. The curves correspond to startup cases where the number of faulty nodes are zero, one, and two.

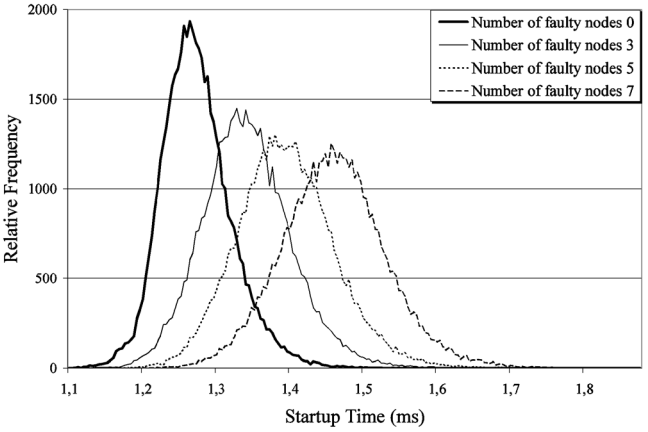


Fig. 12. The relative frequency of the startup times in a 24-node system. The cycle-time is 0.82 ms. The curves correspond to startup cases where the number of fault nodes are varied between zero and seven.

normal operation such that nodes can resynchronize after a transient failure.

The basic TTP/C startup behavior can, under normal conditions, be described in three steps as follows [20]:

1. When the nodes are turned on, they enter an *Init* mode. In the *Init* mode, the nodes run the initialization code.
2. After a node has initialized itself, it enters the *Listen State* where it starts a *listen-timeout* and waits for an initialization message, i.e., the I-frame. If a node receives an I-frame before the timeout it can synchronize itself to the sending node. After the reception of the I-frame, a node transfers to the *Active State*, via the *Ready State*. In the *Active State*, nodes send normal messages, i.e., messages with data information.
3. If a node does not receive the I-frame before the listen-timeout ends, it will send its own I-frame. After sending this I-frame message, the node will wait until it receives a new I-frame message with the same C-state. If such a message is received before the end of the *Cold Start Timeout*, this node will transfer into the *Active* state.

Thus, the node that times out first from the *Listen Timeout* will send the first message, which is an I-frame. Other nodes receiving this message will set its C-State accordingly and can then change to normal operation phase where normal messages can be exchanged, i.e., messages with data information.

The basic startup behavior is, in reality, a bit more complex, for example, TTP/C has a dual communication channel. Such a dual link will affect the startup behavior, but, in order to compare these startup methods, we will compare with a single channel version. In this comparison we are mainly interested in the average startup times of the protocol, but also the max startup times. However, TTP/C has one extremely unlikely startup case where all nodes enter the cold start mode simultaneously. This will generate a very long startup time, but it occurs with very small probability. We have chosen not to simulate such a case, as it is very unlikely and it would affect the average time

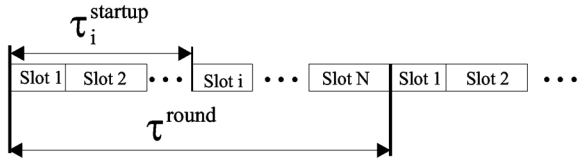


Fig. 13. The startup timeout.

startup minimally. If such a case would occur, several TDMA-rounds would be added to the startup time.

In this simulation, we have assumed that the TTP/C nodes are started approximately at the same time and then they run their initialization code. As a consequence, the *Listen Timeouts* of the nodes are started at different times. In this simulation, we have therefore assumed that nodes start their individual *Listen Timeouts* at different times, evenly distributed between time zero and a cycle time, e.g., for a six node system between 0 and 0.2 ms.

The startup time and behavior of the TTP/C protocol is mainly decided by the startup timeout's $\tau_i^{startup}$, shown in Fig. 13, which is unique to each node.

Together with the TDMA round time τ^{round}, they build the *Cold Start Timeout* and the *Listen Timeout* as follows:

$$\tau_i^{coldstart} = \tau^{round} + \tau_i^{startup}, \quad (7)$$

$$\tau_i^{listen} = 2 \cdot \tau^{round} + \tau_i^{startup}. \quad (8)$$

In Table 4, we can see the average startup times with their standard deviation as well as the Max startup times for the TTP/C protocol compared to the presented ML- approach.

As seen from Table 4, our ML approach compares well when considering average startup time. It should also be emphasized that the ML-approach does not require any special messages to be synchronized, i.e., it uses normal messages. This means that the communication is established very fast using the ML-approach. When the nodes in the ML-approach reach the Normal state (when we have stopped measuring the time of this approach), half of these nodes have already sent messages with data. This is not the case for the TTP/C protocol, which we stopped the time measuring after the first correctly sent message, i.e., the first I-frame.

The DACAPO protocol [18] is a TDMA protocol where all nodes have static and equal lengths, otherwise, it is similar to our approach. When comparing with the DACAPO protocol startup, we have used data from [19] where three startup methods are compared. We will not go into details of these, but note that two of them are developed by one of the authors of this paper. These startup methods are referred to as the ZF and I/D methods. In order to translate those results to our configuration, we have adapted the message length to be $1/6$ of the cycle time. We have compared our case with 6-nodes and a cycle time of 204.8 μs. This means that we have adapted the result from [18] to use a message length of 83 bits and interframe gaps of 2 bits, which results in a slightly smaller cycle time of 204.0 μs. The results can be studied in Table 5. Our primary interest has been in comparing the max startup times to see whether they have improved compared to our initially described startup method. As shown in the table, we have managed to improve the startup times considerably from our previous generation of startup-protocols.

TABLE 4
Average Startup Times for TTP/C Protocol Compared to the Presented Message Length (ML) Approach

Nodes	**CT** (ms)		**TTP/C** (ms)	**ML** (ms)
6	0.20	Average	0.52	0.37
		Std. Dev.	0.05	0.03
		Max.	1.11	0.55
24	0.82	Average	1.85	1.29
		Std. Dev.	0.12	0.05
		Max.	3.81	1.6

TABLE 5
Max Startup Times for ZF, I/D, and LS Startup Methods (Fault-Free Cases) Compared to the Presented Message Length (ML) Approach

Startup Method	**Max** (ms)
ML	0.55
ZF	2.28
I/D	1.80

8 Summary and Conclusions

In this paper, we have presented a unique synchronization approach that provides for low-cost fault-tolerant synchronization in distributed real-time systems targeting safety-critical systems for the mass-market. The proposed TDMA communication approach uses 1) the static information of the different message lengths to obtain information about sender **id** and 2) unique sequences of message lengths to discern ordering. This simple approach provides low-complexity and high-efficiency start-up synchronization and resynchronization. Fundamentally, our approach makes the following contributions:

- Provision of bounded start-up time using a novel deterministic backoff strategy.
- Quick start-up on average, with a small standard deviation.
- Low communication overhead compared to existing TDMA-based approaches.
- Fast reintegration of recovering nodes.
- High robustness in tolerating faults with only a limited synchronization and resynchronization time penalty.

Thus, the synchronization approach appears well suited for real-time control applications due to its guaranteed temporal upper bound on start-up. Its low complexity combined with the simplicity makes it useful for cost-sensitive safety critical systems as well.

Acknowledgments

The authors would like to thank the reviewers of this paper as well as their colleagues Arshad Jhumka, Martin Hiller, and Andréas Johansson for their time and many helpful comments. This work was partly funded by the Volvo Research Foundation #F99/07 and EU Next TTA #IST-2001-32111.

REFERENCES

[1] V. Claesson, H. Lönn, and N. Suri, "Efficient TDMA Synchronization for Distributed Embedded Systems," *Proc. 20th Symp. Reliable Distributed Systems,* pp. 198-201, Oct. 2001.

[2] CAN Specification Version 2.0, Robert Bosch GmbH, 1991.

[3] K.W. Tindell and A. Burns, "Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Net," Dept. of Computer Science, Real-Time Systems Research Group, Univ. of York, Technical Report YCS 229, 1994.

[4] Multi-Transmitter Data Bus, Part 1, Technical Description, Aeronautical Radio, Inc., Dec. 1995.

[5] H. Kopetz, "Should Responsive Systems be Event-Triggered or Time Triggered?" *IEICE Trans. Information and Systems,* vol. E76D, no. 11, pp. 1325-1332, 1993.

[6] H. Sivencrona, L.-Å Johannson, and V. Claesson, "A Novel Bit-Oriented Communication Concept for Distributed Real-Time Systems, qrcontrol," *Proc. Third Int'l Conf. Control and Diagnostics in Automotive Applications,* 2001.

[7] J. Berwanger, M. Peller, and R. Griessbach, "Byteflight—A New High-Performance Data Bus System for Safety-Related Applications," http://www.byteflight.com/, 2004.

[8] P.J. Koopman and B.P. Upender, "Time Division Multiple Access without a Bus Master," United Technologies Research Center, US, Technical Report RR-9500470, 1995.

[9] R.M. Metcalfe and D.R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Comm. ACM,* vol. 19, no. 7, pp. 395-404, 1976.

[10] H. Kopetz and G. Grunsteidl, "TTP—A Protocol for Fault-Tolerant Real-Time Systems," *Computer,* vol. 27, no. 1, pp. 14-23, 1994.

[11] H. Kopetz, A. Krüger, R. Hexel, D. Millinger, R. Nossal, R. Pallierer, and C. Temple, "Redundancy Management in the Time-Triggered Protocol," Technical Univ. of Vienna, Technical Report 4/1996, 1996.

[12] H. Lönn and R. Snedsbøl, "Synchronisation in Safety-Critical Distributed Control Systems," *Proc. IEEE Int'l Conf. Algorithms and Architectures for Parallel Processing,* vol. 2, pp. 891-899, 1995.

[13] N. Suri, M.M. Hugue, and C.J. Walter, "Synchronization Issues in Real-Time Systems," *Proc. IEEE,* vol. 82, no. 1, pp. 41-54, 1994.

[14] P. Ramanathan, K.G. Shin, and R.W. Butler, "Fault-Tolerant Clock Synchronization in Distributed Systems," *Computer,* vol. 23, no. 10, pp. 33-42, Oct. 1990.

[15] H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real Time Systems," *IEEE Trans. Computers,* vol. 36, no. 8, pp. 933-940, Aug. 1987

[16] P. Folkesson, "Assessment and Comparison of Physical Fault Injection Techniques," PhD dissertation, Chalmers Univ. of Technology, 1999.

[17] J. Berwanger, C. Ebner, A. Schedl, R. Belschner, S. Fluhrer, P. Lohrmann, E. Fuchs, D. Millinger, M. Sprachmann, F. Bogenberger, G. Hay, A. Krüger, M. Rausch, W.O. Budde, P. Fuhrmann, and R. Mores, "FlexRay—The Communication System for Advanced Automotive Control Systems," SAE 2001 World Congress, ser. SAE Technical Paper Series, Detroit, Mich., 2001.

[18] B. Rostamzadeh, H. Lönn, R. Snedsbøl, and J. Torin, "DACAPO: A Distributed Computer Architecture for Safety-Critical Control Applications," *Proc. Intelligent Vehicles Symp.,* pp. 376-381, 1995.

[19] H. Lönn, "Initial Synchronization of TDMA Communication in Distributed Real-Time System," *Proc. 19th IEEE Int'l Conf. Distributed Computing Systems,* pp. 370-379, 1999.

[20] *TTP/C Protocol, Specification of the Basic TTP/C Protocol,* first ed., Time-Triggered Technology, TTTech Computertechnik GmbH, www.tttech.com, July 1999.

[21] W. Steiner and M. Paulitsch, "The Transition from Asynchronous to Synchronous System Operation: An Approach for Distributed Fault-Tolerant Systems," *Proc. Int'l Conf. Distributed Computing Systems,* pp. 329-336, July 2002.

Vilgot Claesson received the PhD degree from Chalmers University of Technology, Sweden. He is currently a research fellow at TU Darmstadt, Germany, and is associated with Volvo Technology serving a liaison role across academic and industrial research. Previously, he worked with system and software design on the AXE10 group switch at Ericsson Research and Development. His research interests are in real-time distributed embedded systems. Currently, the focus is on time-triggered systems and their extension to provide efficient event-triggered services, such that they meet the dependability and functionality demands of aerospace-and automotive arenas. He is a member of the IEEE and the IEEE Computer Society.

Henrik Lönn received the PhD degree in computer engineering from Chalmers University of Technology, Sweden. He has researched communication issues in real-time systems with a special focus on initialization and clock synchronization. He is currently at Volvo Technology Corporation, Department of Electronics and Software. At Volvo, he has worked with prototypes, architecture modeling, and communication aspects on vehicle electronic systems. He is also participating in national and international research collaborations on embedded systems development. He is a member of the IEEE and the IEEE Computer Society.

Neeraj Suri received the PhD degree from the University of Massachusetts at Amherst. He currently holds the TU Darmstadt chair professorship in "Dependable Embedded Systems and Software" at TU Darmstadt, Germany, and is also affiliated with the University of Texas at Austin. His earlier academic appointments include the Saab Endowed Professorship and, earlier, at Boston University. His research interests focus on design, analysis, and assessment of dependable embedded systems and software. His current research is emphasizing robustness hardening of software, verification along with experimental validation of protocols, embedded software and operating systems, and "security by design" for SW and OSs. His group's research activities have garnered support from DARPA, the US National Science Foundation, ONR, European Commission, NASA, Boeing, Microsoft, Intel, Saab, Volvo, and Daimler Chrysler among others. He is also a recipient of the US National Science Foundation CAREER award. He serves as an editor for *ACM Computing Surveys* covering embedded systems and real-time, and has been an editor for the *IEEE Transactions on Parallel and Distributed Systems*. He is a member of IFIP WG 10.4 on Dependability, a senior member of the IEEE and IEEE Computer Society, and also on the board for Microsoft's Trustworthy Computing Academic Advisory Board. More professional details are available at: http://www.deeds.informatik.tu-darmstadt.de/suri/activities/activities.html.