An Efficient TDMA Synchronization Approach for Distributed Embedded Systems

Vilgot Claesson*

Dep. of Comp. Engineering
Chalmers University
S-412 96 Göteborg

Henrik Lönn

Volvo Technological Development
Electronics and Software
S-412 88 Göteborg

Neeraj Suri

Dep. of Comp. Engineering
Chalmers University
S-412 96 Göteborg

Abstract

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

Contact author: Vilgot Claesson

Email: vilgotc@ce.chalmers.se

Keywords: Distributed systems, Real-Time, Synchronization, TDMA, Startup, Protocol, communication.

Submission category: Regular paper.

* Supported by Volvo Research Foundation, Proj. F99/7

1 Background and Motivation

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 meeting a cost-sensitive mass-market safety critical system (e.g., computer control), we develop low-cost and fault-tolerant synchronization strategies in Time Division Multiple Access (TDMA) bus environments.

In this paper, we investigate low-cost, highly efficient, and fault-tolerant communication primitives for safety-critical systems with hard real-time requirements. 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 as a form of message addressing on which a synchronization approach can be built. A solution for avoiding start up collisions is also discussed using a similar method.

Bus systems are prolifically used in many systems today on account of the simplicity and low cost. In computer control, many different protocols utilize a communication bus, for example, CAN [9]. However, with a communication bus a number of different media access strategies exist on how a node can potentially access the bus. The most common way to solve this is by contention resolution, more specifically Carrier Sense Multiple Access (CSMA), where each node senses the bus for activity and may send when none is detected. This has the drawback that if two 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 lack of determinism. With CSMA/CD, there is no way to deterministically avoid repeated collisions, which effectively constrains estimating worst-case communication times.

To avoid the limitations with CSMA/CD for real-time systems, Collision Avoidance can be used, as for example, in CAN. Collision Avoidance using bit arbitration is termed as CSMA/CA. 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 [12]. Using bit arbitration, the bus propagation time requires 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 [1] are other fundamentally different medium access schemes. However, 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 pre-assigned timeslots in a repeating schedule. TDMA communication provides a very 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 inherited from the determinism of TDMA communication are very useful, for example, evident timing, composability, easy fault detection, and testing.

Furthermore, most computer control systems have real-time requirements where these properties are particularly important, especially when combined with the safety requirements, i.e., hard real-time system, where the consequences are catastrophic if deadlines are missed.

With TDMA communication, the nodes must be synchronized in order for each node to know when to send and receive data on the bus. At this point we identify two problems with TDMA synchronization. First, we require routines to handle the communication start-up such that nodes get synchronized fast and within a short period of time. Second, when all nodes are synchronized, it is sufficient to uphold the bit synchronization in order for nodes to know when to send their messages. Furthermore, since the communication schedule is static and done pre-runtime, each node has the knowledge of which node is the sender of each message. However, in the case of a transient fault that causes the node to lose synchronization, this node will not know the position in the TDMA communication. This makes it impossible for the node to ascertain its message sending time. Therefore, it is necessary that some node(s) send this information periodically for nodes that need to resynchronize.

2 Basic Approach and Paper Organization

System and clock initialization is a necessary component in safety critical applications for nodes to get their clocks synchronized, both at start-up and after error recovery. However, at start-up there is a possibility that nodes that send their initial time message will collide, consequently withdrawing and attempting a later retry. If, at their next try, they collide again, the system can potentially end up in a state with an infinite series of collisions. Thus, the initialization process must prevent such sequences of collisions. In our proposed synchronization approach, we prevent this by providing each node with a unique adjustment time. This time is used to adjust the local clock of a node in case of collision. By doing so, the time to the new send retry is increased, and we refer to this time as the time increment. The scenarios for which this approach resolves collision sequences are presented in this paper.

To get nodes synchronized to the common communication schedule, we let the node that first successfully sends a message distribute necessary timing information to the remaining nodes. This timing information is normally done by special messages or is included in normal messages, both resulting in communication overhead. In TDMA communication with a static schedule, we normally know from the sender's **id**, the position in the schedule. So, the first sending node can provide the senders **id** in the sent message instead of distributing the time.

Our approach proposes to take this a step further, where nodes use the knowledge of the message lengths to find out the position in the communication schedule. Each node sends messages with only one message length, and this message length is unique for this node. With this simple approach, a node can determine the sender's **id** by the length of the received message. This provides the basis for our synchronization approach, which becomes both efficient and robust, since no additional time/**id** information or special messages need to be sent. The approach

aids determining of the communication mode in progress, as well.

This paper is organized as follows. In Section 3, we review related work. This is followed by a presentation of the system and fault model in Section 4. In the main Section 5, the initialization and resynchronization approaches are described in more detail, followed by an evaluation and a proof of the upper bound on startup time. Finally, a summary and conclusion is given.

3 Related Work

Several solutions to TDMA system start-up exist at present, though most deal with systems of limited size where the propagation delay is easily bounded. One possible approach is to use a known bit-pattern at the beginning of each frame or TDMA cycle [4, 11]. If a unique bit pattern is used, synchronization is obtained when this pattern is detected. If it is not unique, the node can search the transmitted bits for maximum correlation with the known pattern and, eventually, obtain synchronization. Both of these approaches have certain drawbacks for the type of safety-critical applications of interest for us. Also, if a unique bit-pattern is used, stuffing bits or the like are necessary to avoid this bit pattern within ordinary messages; in embedded systems for control applications, messages are typically short, in the 100-bits range, and any extra bits result in a large overhead. The second approach implies that maximum correlation must be found, which is complex and adds time overhead. Instead of a bit-pattern, a unique signal level can be used. However, the extra hardware necessary would probably be more efficiently used to improve the bit encoding. Also, if a faulty node repeatedly issues the resynchronization signal, such a failure would be more severe and difficult to mask than other failure modes that result in invalid transmissions. If, instead, 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 correct.

In the TDMA protocol TTP [5], 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 initialization mode is entered. Moreover, if collisions are detected while in this mode, all nodes must reset their clocks [6]. In the Lightning architecture [2], the lockstep method is used. During start-up, a node is not allowed to send until its predecessor has transmitted, except for a dedicated ”first” node. A time-out is used to detect node failures and allow nodes to start sending even if their predecessor has failed.

4 System, Communication, and Fault Models Behind the Approach

The System Model: The systems of interest are safety-critical systems 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 nodes operate in a cyclic manner, where

each node sends only once in each communication round, i.e., the TDMA-round, as shown in Figure 1. Each node i may send messages of different length denoted by tm_i. To increase the flexibility compared to a fully static communication schedule where message order and length are fixed, the system can use different communication modes. In each mode, the nodes may use a different length for their messages, which will be fixed within this mode. Hence, in each communication mode the length of the messages, and thus the TDMA round, are static.

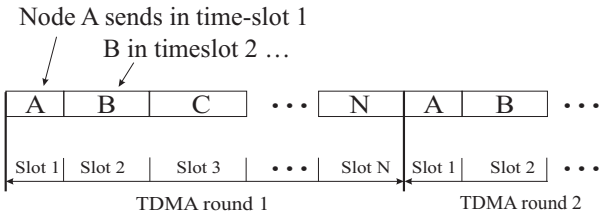

Figure 1: A TDMA communication round.

If the system runs in a given mode, e.g., operational mode or start-up mode, the communication system will run in a corresponding communication mode. These communication modes are termed as system cycles and each consists of a number of TDMA-rounds. In these system cycles, the number of TDMA-rounds, the message content and message length are specified. A system-cycle is repeated periodically. Thus, each node can change the contents of its message for each TDMA-round in the system-cycle. A system-cycle is also static and the contents of messages in a system-cycle are known. However, a system-cycle will not change the basic behavior of the TDMA-round and we do not consider them further here.

Communication Model: When the system is starting up, the local clocks of the nodes are not synchronized. Thus, nodes do not possess global knowledge on when to send messages as no global time has been established. The purpose of a start-up mechanism in a TDMA system is to get the nodes to agree on the time, i.e., achieve a global time-base. Depending on the system requirements, it may be very important that the startup of communication is fast. For example, after a complete system failure, e.g. power outage etc., where the synchronization is lost, it is important that the system is reinitialized quickly within the systems stipulated response time.

We consider synchronization to be divided into two logical levels, (1) synchronization on frame level and (2) synchronization on the communication schedule level. We assume that the frame level synchronization is handled by a standard clock synchronization algorithm, for example, the daisy-chain clock synchronization algorithm [7]. The frame level synchronization ensures that each bit-value is received correct. The purpose of synchronization at the communication schedule level (2) is to synchronize the nodes to the same phase of the communication schedule. This ensures that every node has the same view of the current position in the communication schedule. With the knowledge of this position together with the local clock, the nodes will know when to send their messages. Thus, the necessary information for a node is the global time, which consists of fine-grained time ensuring correct bit values and the coarse-grained schedule time. Each node must store the communication schedule, which will

contain the information of when and what a node should send.

The global time-base can be used to synchronize distributed task and minimize delay and jitter, by relating execution times of tasks to the TDMA rounds. This eases 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 therefore to keep the primitives as simple as possible. Communication bandwidth for embedded system is sparse and has developed very slowly when compared to the growth in processing efficiency. It is therefore very important to have efficient protocols with limited communication overhead.

The Fault Model: The initial synchronization algorithm require 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 $\lfloor (n-1)/2 \rfloor$. When using a broadcast bus we require that nodes will follow a fail-silent semantic, such that no faulty node fails by continuously transmitting on the bus. Such a failure would overflow the bus and prevent any normal communication, including any synchronization effort. The fail-silent property relies on the coverage of error detection mechanisms used in the node. It can be argued that such a fault semantic is not applicable in a hard real-time system, since sufficient coverage is not possible to achieve. However, recent work indicates that using rigorous design and error detection method, a very high coverage can be achieved [3].

For the bus, we assume an omission failure semantic where messages are either received correctly and on time or not at all. By designing nodes to be fail silent and using a bus system efficiently exclude timing failure on the bus. Similarly, Byzantine failures are not considered as they are easily avoided by design, using a bus combined with message checksums.

The initial synchronization will thus tolerate $\lfloor (n-1)/2 \rfloor$ node failures and message omissions. The number of message omissions may affect the synchronization time, which will be discussed in Section 5.2. A node recovering from a failure will need resynchronization in order to send messages. A node using this synchronization method will immediately regain synchronization after the first correct message reception. Thus, assuming normal operations, message omissions will only affect the time to resynchronization of the node.

In the next Section we discuss our approach together with current approaches on initial and resynchronization, highlighting properties and shortcomings, to put our approach in perspective.

5 Initialization and Resynchronizations: Outlining the Basic Approach

The initialization problem of TDMA communication is that nodes can only be synchronized by sending messages on the broadcast bus. However, to be able to send messages on the bus such that collisions are avoided, the nodes must be synchronized in the first place. This leads to a situation where, similar to CSMA/CD, no upper bound can be put on the initialization. For real-time systems and especially hard real-time systems, a more deterministic startup is desirable.

Current Approaches: The usual way to exchange time and thus the position in the communication schedule is to send the time of local clocks explicitly in messages. An 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. This can be done since messages are pre-scheduled and these time values will reflect the local time of the corresponding sender. That is, we extract the sender's opinion of the time when it sends its message according to the "sender clock". For this to work, we must first be able to identify the sender of the message and then the static schedule provides the send time. The difference from the scheduled send time and the time when the message was actually received is used to create a correction of the receivers clock to the senders clock.

A message identifier is usually included in the beginning of messages, as **id**-fields. However, when using static scheduled messages, the reception time of the message can serve as the **id**, thus making the **id** field of messages unnecessary under normal operation. This assumes that nodes have synchronized their local clocks, i.e., the nodes have agreed on the position in the communication schedule. Thus, before the nodes are synchronized or when nodes lose synchrony, explicit message **id** is necessary. Thus, nodes need to send explicit message **id**'s until nodes are synchronized.

To handle initial synchronization and resynchronization we can send the message-**id** in all messages or send special initial messages as done in DACAPO [10] and TTP [5] respectively. However, sending the **id** in all messages will add extra overhead, which is only useful at startup and at synchronization. Thus, sending special messages at startup appears as a good idea. However, then we automatically add an extra communication mode, which adds to the complexity. Furthermore, resynchronization of nodes is not handled, thus, an extra mode must be added to which the system changes if one node loses synchronization. In such a case a membership service must be introduced. 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 lost synchronization will not be able to reintegrate. Therefore, additional nodes have to send messages with resynchronization information in order to tolerate any failures. The disadvantage is the requirement of an extra mode and the additional overhead at runtime.

Proposed Approach: Our solution is to take advantage of information that is inherent in statically scheduled communication. Instead of including the message **id** in each message, we use the length of messages as the **id**. This basically implies that all messages have different lengths and that the length of the time-slots correspond to these message lengths. With this knowledge, a message receiver immediately knows who the sender is by using the message length information. However, this requires that each node possesses a list of the message lengths and to which node they correspond. The unique message lengths of nodes are chosen according to the required bandwidth of each node, such that the node with most data to transfer (per time unit) will have the largest message. In the case where nodes require the same message length we have to break the symmetry by differentiate the message length of these nodes. This approach is very useful both during start-up and resynchronization, as a node will require only one message receipt to be able to achieve system-level synchronization.

Using this approach, nodes will know the position of the communication schedule as soon as one message 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. The usual workaround for this is to wait a random delay before resending, usually called exponential back-off[1]. This usually works but for hard real-time systems a bounded time on startup and resynchronization is required.

To achieve a bounded start-up when messages collide, each node will delay its retry by a unique time period. We will show that, using time periods that are short compared to the cycle time, the worst-case start-up time is bounded. This worst case scenario is, however, extremely unlikely and normal startup times will be shown via simulations.

In most systems, the nodes are likely to have different requirements on bandwidth, thus the requirement of unique message length is normally not a limitation. It offers more flexibility than a system using messages of the same length. The requirement of unique message length can even be relaxed if some nodes need to send the same amount of data. Nodes can then use messages with the same length by adding simple constraints on sending at the initial synchronization. Such a constraint could, for example, be that only messages longer than previously sent messages are allowed to be sent. This simple algorithm will allow many messages with the same message length. Although it is easy to come up with communication schedules for which the suggested algorithm would not work, there are in most cases other simple algorithms which will work. The simplest solution to solve the problem is however to make a message slightly longer, i.e., add padded bits.

To give an example where we have many short messages requiring similar amount of sent data, it might be difficult to choose different lengths for each node. This situation can easily be handled with two adaptations. (1) At startup, only nodes with uniquely identifiable messages lengths are allowed to send. This means that during startup the received messages must be identifiable, such that nodes can synchronize to them. In Figure 2, we show an example where only nodes with odd **id** numbers would be allowed to send. (2) During resynchronizations, at least a unique sequence of lengths is required, such that a reintegrating node can distinguish a unique message pattern and thereby locate its position in the TDMA cycle. In the example in Figure 2, a resynchronizing node can be synchronized as soon as a message from a node with odd **id** is received. Thus, for startup, only nodes with unique message lengths can send, such that the startup is fast. For reintegration, it is sufficient with a unique sequence of message lengths.

5.1 The Node-Level Synchronization Operations

On this background, we now describe in detail the node synchronization process. Nodes work in three different modes depending on the level of synchronization achieved. 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

[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.

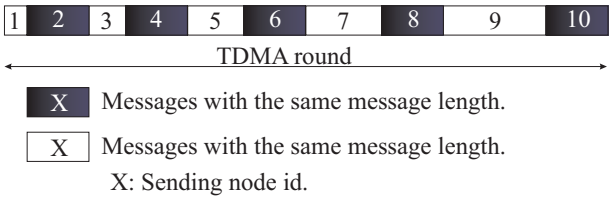

Figure 2: A TDMA communication round, with nodes with odd **id** numbers having unique message length. Nodes with even **id** numbers all have the same message length.

lost synchronization or messages from less than half of the nodes are received. Mode (3) is the *recovery mode* that a node enters at startup and after a disturbance preventing message reception for the duration of a full TDMA cycle. The rules are based on those described by H. Lönn in [8]. We first provide some relevant definitions that are used in the synchronization protocol description in Section 5.1.2.

5.1.1 Definitions

$\mathbf{tm_i}$ 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 alloted a unique time period INC_i used to delay the nodes transmission after a collision.

$\mathbf{tm_{max}}$ 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.

MRvec Message Receive Vector, containing on information whether the latest n messages were received correctly or not. In a system with eight node's ($n = 8$) where a nodes $MRvec$ contains three 0's and five 1's, $\{$0,1,0,0,1,1,1,1$\}$ means that this node has received five messages correctly and three incorrectly. $MRvec$ is a FIFO list where a new message is inserted and the oldest are shifted out. In case of an incorrect message or message omission a 0's is shifted into the vector, otherwise, if a correct message is received a 1 is put in the vector. We denote the number of 1's in $MRvec$ with $ones(MRvec)$.

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

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

5.1.2 Synchronization protocol, modes and operation

The following operations are performed locally at each node. A node will also treat message receptions from itself in the same way as other nodes, e.g., updating the $MRvec$.

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

1. For each correctly received message M_s sent by node s, the receiving node r updates the time counter TC_r by adding the time tm_s corresponding to the received message M_s. Thus $TC_r = TC_r + tm_s$. The Received message vector $MRvec$ is updated, a 1 is shifted into the vector.
2. If an incorrect or no message was received by a node r, the time counter TC_r of node r is incremented with the duration tm_i of the expected message. The Received message vector $MRvec$ is updated, a 0 is shifted into the vector.
3. Enter Resynchronization mode if messages from less than half of the nodes have been received correctly, i.e., number of ones in $MRvec$ is less than half of the number of nodes, $ones(MRvec) \leq \lfloor n/2 \rfloor$.
4. A node i will send its message when the time counter is equal to the send time, $TC_i = TS_i$.

Resynchronization mode Having sent a message in recover mode, a node enters resynchronization mode and waits for reception of messages from half of the nodes. (Note: Nodes only listen for messages on the bus):

1. When a correct message M_s is received by node r, set the time counter TC_r according to the end time of this message, $TC_r = ST_s + tm_s$. The Received message vector $MRvec$ is updated, a 1 is shifted into the vector.
2. If an incorrect or no message was received by a node r, the time counter TC_r of node r is incremented with the sending time tm_i of the expected message. The Received message vector $MRvec$ is updated, a 0 is shifted into the vector.
3. Enter Normal mode when messages from a majority of nodes have been received correctly ($ones(MRvec) > \lfloor n/2 \rfloor$).
4. If the bus has been completely silent for one communication cycle (silence counter is greater then the time period, $SC >= T$) enter recovery mode.

Recovery mode The start mode, where nodes return in case of complete loss of synchronization.

1. A node i is allowed to send a message if the following condition are fulfilled:
   - The local time counter TC_i indicates that it is node i's turn to send, we refer to this time as ST_i.
2. When a correct message M_s is received by node r, set the time counter TC_r according to the end time of this message, $TC_r = ST_s + t_{ms}$. The Received message vector $MRvec$ is updated, a 1 is shifted into the vector.
3. If incorrect or no message was received by a node r the time counter TC_r of node r is incremented with the duration tm_i of the expected message. The Received message vector $MRvec$ is updated, a 0 is shifted into the vector.
4. If a collision was detected by node i when sending its message, it will immediately stop sending and postpone its next intended time to send, with the time INC_i, i.e., next transmit time is $INC_i + T$, where T is the period time.
5. When one message has been successfully sent, enter Resynchronization mode.
6. Enter Normal mode when messages from a majority of the nodes ($ones(MRvec > \lfloor n/2 \rfloor$) have been received correctly. in $MRvec$.

In all these modes, we require message receipt from a majority of the nodes, thus we tolerate $\lfloor (n-1)/2 \rfloor$ faulty nodes. The effect of failure both during start-up and resynchronization of a node is penalty in time. This will be shown in our simulation results.

It is important to avoid repeated collisions of messages which would prevent the system from synchronizing. This is handled in the recovery mode above. The idea is that each node i is assigned a time increment INC_i, significantly shorter than the period time T. If a collision occurs, colliding nodes will postpone their next retry

with a time equal to the increment INC_i. Thus, the next send time t_{k+1} for node i will thus be one period plus the time increment later, i.e., $t_{k+1} = t_k + T + INC_i$. This will postpone the resending of the message within the time period INC_i. We summarize the initial algorithm in the list below where we assume that none of the nodes is synchronized:

- All nodes start in *Recover mode* described above.
- If a node i sends a message and other nodes receive it correctly, the receivers are assumed to synchronize to node i's clock, the system is thereby synchronized.
- Each node i is assigned a unique time period INC_i, and $INC_i << T$, where T is the period time of a TDMA cycle.
- Each node i sends at the pre-assigned time-slot according to the time counter TC_i.
- If node i's message collides, it will add INC_i to its time counter TC_i prior to resending.

If further collisions occur, a node will keep adding its unique time period INC_i to its time counter. After a deterministic time interval as detailed in Section 5.3, Equation (6), at least one message will not collide. All nodes will receive this message and synchronize to it, which will prevent further collisions. Thereby, the initial synchronization completes.

In the worst case scenario the number of collisions will be $\lfloor n/2 \rfloor$, where n is the number of nodes in the system. In Section 5.3 we will derive this figure and show its correctness.

5.2 Properties and simulations

In this section, we discuss and show via simulation results the properties of this synchronization approach. The analytical proof of the bounded startup time appears in Section 5.3. 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 transfered in order to achieve synchronization. The time overhead relates to the additional timeouts, retransmissions etc. Finally the computation/memory 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 limited overhead. However, there is invariably some overhead related to communication services like synchronization. In safety critical real-time systems, the synchronization time 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.
- **Time:** Synchronization is accomplished as soon as a message is received. It may be delayed by collisions and faulty nodes, bounded as given in Equation (6), Section 5.3.
- **Computation/Memory:**
  - Each node carries a list of "n" entries containing all message lengths and the corresponding nodes, it also include the knowledge about the time slot it should be sent out in.
  - A counter which keeps track of local time, i.e., the point of the communication schedule the system has currently reached.
  - 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 this proposed Message Length (ML) based approach, we compare it with two existing approaches for initial synchronization and resynchronization. This approach used in DACAPO [10] (*D-approach*) 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 one used in TTP [5], (*T-approach*), where nodes send special start frames and each node uses dedicated back-off times. Finally, we include an approach that behaves as ML except that it uses exponential back-off instead of time increments 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 increase the time range from which the random delay is chosen. The exponential back-off is used in, for example, the Ethernet. This method would require a random number generator in each node instead of our increment value stored pre-runtime. 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 (a) communication bandwidth overhead (in # of bits), (b) storage overhead, (c) bounded/unbounded startup time and (d) maximum time for a node to resynchronize. In our ML approach and ML_{exp}, there are no communication overhead, 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 startup time, see Section 5.3.

For the resynchronization approach, the time to resynchronize a recovered node is totally dependent on the time of reception of a first message. The time to send messages from node i is tm_i and we assume that we index the nodes such that the time slot 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 would be:

$$t_{resync} = tm_{n-1} + tm_n \tag{1}$$

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

$$t_{resync} = tm_{n-1-f} + \cdots + tm_{n-1} + tm_n \quad (2)$$

As we see in Equation (2), t_{resync} 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.

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 these types of messages. 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.

The start-up time for this approach is normally bounded, since all nodes should listen to the bus and reset their local clock at this time. 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 be sure of avoiding additional collisions, all nodes must have sensed the collision. By increasing the time between two successive initialization messages, we can save the bandwidth for additional payload. This will however increase the worst-case time for resynchronization of a recovered node. The worst-case resynchronization time depends on the time between and 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.

The D-approach always sends *sender id* in the message and will get 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 startup. 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 startup synchronization and resynchronization.

Table 1 outlines that the ML-approach combines a bounded start-up time with low communication overhead and fast resynchronization.

To show the general startup behavior of our approach, we have simulated the initial startup synchronization and measured the time for all nodes to reach the Normal mode. We have assumed that we have 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 our communication bits are longer such that nodes detect all collisions. In these simulations, the basic time-units are 0.4 μs, i.e., the basic bit transmission time. We simulate the startup for two system sizes, 6 and 12 nodes, for the 6-node-case we simulate with 0, 1 and 2 faults.

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 Figure 3. We have assumed that message lengths have been in the order of existing communication protocols, e.g., CAN with 0-64 bits of data. Thus we have chosen message length between 24-80 bits for our simulation. The message lengths are unique and chosen as a multiple of the basic time unit, of 0.4μs. The time increments (INC_i) are also multiples of the basic time unit, and chosen such that they

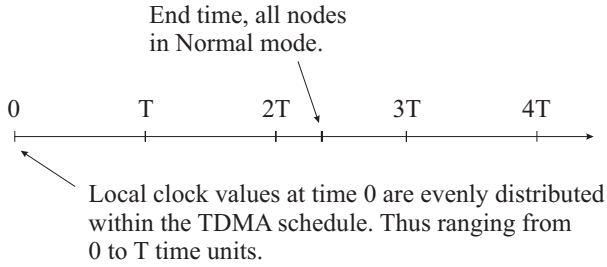

Figure 3: Startup initialization.

Method	Comm.(bits)	Memory (bits)	Startup	Resynch. (secs)
Message Length (ML)	0	$\propto n$	Bounded	$(f+1) \cdot tm_{max}$
D-Approach	$log_2(n) \cdot n$	$log_2(n)$	Bounded	$(f+1) \cdot tm$
T-Approach	$log_2(n) \cdot (f+1)$	$log_2(n)$	Bounded	$f \cdot t_d + tm_{im}$
ML_{exp}. back-off	0	$\propto n$	Unbounded	$(f+1) \cdot tm$

Table 1: Properties of startup/resynchronisation approaches.

are much smaller than the message lengths. For our first simulation with 6 nodes, we have chosen the following message lengths (24, 40, 48, 56, 64, 80) time units, giving a TDMA-cycle time of $T = 312$ time units or $124.8\mu s$. The time increments are approximately one tenth of the message lengths. For the second simulation with 12 nodes the message lengths (24, 30, 40, 44, 48, 51, 56, 59, 64, 71, 80, 85) time units have been used which gives $T = 652$ time units or $260.8\mu s$.

Figure 4 and Table 2 show the start-up times for the different start-up simulations. They are based on 1000 start-ups per system configuration, where the local clocks have been assigned random times, according to Figure 3.

The start-up simulations have been performed to obtain the expected start-up time and how the startup behaves in the case of faulty nodes. In Figure 4, the start-up time distribution is shown for the different simulations. The number of start-ups finishing within a specific time interval are plotted against the total number of start-ups per simulation case. The case with 12 nodes takes longer time to start-up than a system with 6 nodes. We also see how the start-up times are increased and diffused when one or two nodes have crashed.

The mean, standard deviation, minimum, and maximum of the start-up times for the simulations are shown in Table 2. In the average case for the 12 nodes system, all nodes are in Normal operation after 1.3 TDMA-cycles, i.e., 353 μs. For the 6 nodes case all nodes are in Normal mode after 212 μs in average, i.e., 1.7 TDMA-cycles. In a run of 8000 start-ups there were 65 start-ups with collisions, and of those 65, there were two with double collisions. Table 2 summarizes these results.

It is important to note that *these start-up times are the time from start until all nodes are in the Normal mode*. All nodes will be in Normal mode when *all* nodes have received messages from more than half of the nodes in the system. This means that at the time we measured, messages from half of the nodes are already received. Since there are no special start-up messages in our approach, useful communication has already begun when all nodes

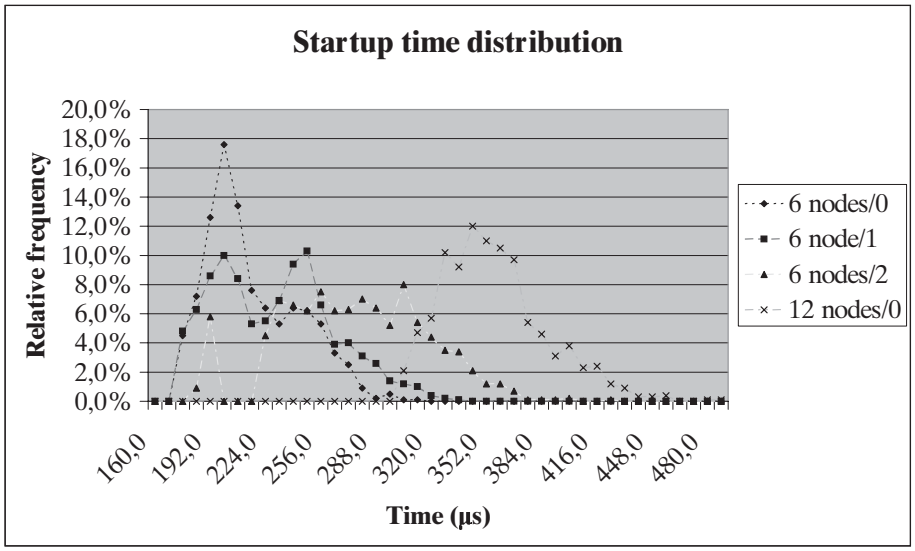

Figure 4: Start-Up distribution from simulations. The label "6 node/X" indicates that 6 nodes are used and X nodes have failed.

No. of nodes/faulty nodes	12/0	6/0	6/1	6/2
Mean time (μs)	353	212	226	271
Std. dev. (μs)	30	27	34	42
Min (μs)	299	170	170	183
Max (μs)	482	320	330	416

Table 2: Start-Up times in time units, for system with 6 and 12 nodes.

are in Normal mode.

It should also be noted that the standard deviation is low; for the 6 and 12 nodes cases the standard deviation is only 13 and 9 percent respectively.

For the Max values we can see that they are within the maximum upper bound, see Equation (6) in Section 5.3, since for the 12-node-case we have, (the maximum increment delay, INC_n, is 5.4μs):

$$482 < (\lfloor 12/2 \rfloor + 1)(T + INC_n) = 7(260 + 5.4) = 1858 \tag{3}$$

given in μs. The start-up time is clearly within the maximum start-up time, since we should remember that the max simulation value is measured when all nodes have reached the Normal mode. The "maximum upper bound" formula from Equation (6) is until one node has successfully sent a message, this is naturally valid for the 6th node case as well. For the 6-node-case the longest start-up time is for the case with two failures, if we make the same comparison, we get (the maximum increment delay, INC_n, is 3.2μs):

$$416 < (\lfloor 6/2 \rfloor + 1)(T + INC_n) = 4(124 + 3.2) = 512 \tag{4}$$

In both cases we should also note that the simulation time is taken from start-up of the system, while the worst-case start-up time is from the first time a node tries to send. This makes both cases well within the limit.

Comparing these figures with other start-up synchronization solutions is not easy, given the multitude of parameters to tune. Similar mean values would be achieved using exponential back-off instead of our increment method, since the ML_{exp} method only behaves differently after collisions.

5.3 Proof of correctness

In this section we will prove the upper bound on subsequent collisions, which is $\lfloor n/2 \rfloor$. In an initial start-up scenario, a node will start in the recover mode. In the recover mode a node will synchronize to the first received message. Thus, we need to show that the nodes, in a bounded time, receive a message to synchronize against.

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

$$INC_1 < INC_2 < \cdots < INC_n \tag{5}$$

where n is the index of the last node. Thus, node n will use the longest delay after a collision.

Then we must prevent a scenario where an infinite sequence of collisions occur on the bus. This is done by postponing a node i's next send retry with INC_i after a collision, whereby preventing the same nodes from colliding at the next try. Using these assumptions, we can draw the following conclusions:

1. To maximize the number of collisions only two nodes can collide at the same time.
2. Nodes with the longest delay must collide first, i.e., node N_n will collide with N_{n-1} and then node N_{n-2} with N_{n-3} and so on.

For each collision, the "fast" node will move forward a step and must collide with a node that has a lower delay. An example with 10 node follows where in each row (N_i,N_j) indicates that nodes N_i and N_j have collided and K indicates the collision number:

$$K = 1 \qquad (N_{10}, N_9)(N_8, N_7)(N_6, N_5)(N_4, N_3)(N_2, N_1)$$

$$K = 2 \qquad (N_9, N_7)(N_{10}, N_5)(N_8, N_3)(N_6, N_1)(N_4, N_2)$$

$$K = 3 \qquad (N_7, N_5)(N_9, N_3)(N_{10}, N_1)(N_8, N_2)(N_6, N_4)$$

$$K = 4 \qquad (N_5, N_3)(N_7, N_1)(N_9, N_2)(N_{10}, N_4)(N_8, N_6)$$

$$K = 5 \qquad (N_3, N_1)(N_5, N_2)(N_7, N_4)(N_9, N_6)(N_{10}, N_8)$$

Thus for n nodes we will get a maximum of $\lfloor n/2 \rfloor$ collisions. An odd number of nodes will only require that we have a number of triple collisions which will reduce the probability of collisions.

The longest time for a message to be sent without collisions is then:

$$t_{startup} = (\lfloor n/2 \rfloor + 1)(T + INC_n) \tag{6}$$

where T is the TDMA round time and INC_n the largest increment among the nodes.

6 Summary and Conclusions

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

- Provision of bounded start-up time.
- 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 systems, 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.

In future research, we plan to conduct additional simulations to obtain performance comparisons with the other synchronization approaches to provide a comprehensive classification of the synchronization schemes as per their overhead costs, efficiency and fault tolerance.

Acknowledgments

The authors would like to thank Arshad Jhumka and Martin Hiller for their time and many helpful comments.

References

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

[2] P. Dowd, J. Perreault, J. Chu, D. Hoffmeister, R. Minnich, D. Burns, F. Hady, Y. Chen, M. Dagenais, and D. Stone. LIGHTNING network and systems architecture. *IEEE/OSA Journal Of Lightwave Technology*, 14(6):1371–1387, 1996.

[3] P. Folkesson. *Assessment and Comparison of Physical Fault Injection Techniques*. PhD thesis, Chalmers University of Technology, 1999.
[4] P. J. Koopman and B. P. Upender. Time division multiple access without a bus master. Technical Report RR-9500470, United Technologies Research Center, USA, 1995.
[5] H. Kopetz and G. Grunsteidl. TTP-a protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, 1994.
[6] H. Kopetz, A. Krüger, R. Hexel, D. Millinger, R. Nossal, R. Pallierer, and C. Temple. Redundancy management in the time-triggered protocol. Technical Report 4/1996, Technical University of Vienna, 1996.
[7] H. Lönn. The fault tolerant daisy chain clock synchronization algorithm. Technical Report 99-2, Department of Computer Engineering, Chalmers University, 1999.
[8] H. Lönn. Initial synchronization of TDMA communication in distributed real-time system. In *19th IEEE International Conference on Distributed Computing Systems (ICDCS'99), Austin, TX, USA*, 1999.
[9] Robert Bosch GmbH. *CAN Specification Version 2.0*, 1991.
[10] B. Rostamzadeh, H. Lönn, R. Snedsbol, and J. Torin. Dacapo: a distributed computer architecture for safety-critical control applications. In *Intelligent Vehicles Symposium*, pages 376 – 381, Detroit, MI, USA, 1995.
[11] W. Stallings. *Data and Computer Communications*. Macmillan Publishing Company, New York, 1991.
[12] K. Tindell and A. Burns. Guaranteed message latencies for distributed safety-critical hard real-time control net. Technical report, Department of computer science, real-time systems research group, University of York, 1994.