

Estimating Message Latencies in Time-Triggered Shared-Clock Scheduling Protocols Built on CAN Network

Mouaaz Nahas*

Department of Electrical Engineering, College of Engineering and Islamic Architecture, Umm Al-Qura University, Makkah, KSA
*Corresponding author: mmnahas@uqu.edu.sa

Received November 23, 2013; Revised December 20, 2013; Accepted January 01, 2014

Abstract The Controller Area Network (CAN) is an event-triggered protocol that is widely used in distributed real-time embedded systems. It has been demonstrated that a “Shared-Clock” (S-C) scheduling protocol can be used on top of CAN hardware to implement time-triggered network operations. Previous work in this area has led to the development of five different time-triggered S-C scheduling protocols referred to as: TTC-SCC1, TTC-SCC2, TTC-SCC3, TTC-SCC4 and TTC-SCC5 schedulers. This paper develops mathematical models for assessing message latencies between all communicating nodes in the different S-C scheduling protocols. In particular, the paper provides mathematical equations for estimating Master-to-Slave, Slave-to-Master, and Slave-to-Slave message latencies in all five schedulers. The paper then presents a small case study to allow a practical comparison of the communication behavior in the various S-C schedulers considered. The results show that the communication behavior, especially Slave-to-Slave message delays, can be improved significantly when TTC-SCC3, TTC-SCC4 and TTC-SCC5 scheduler implementations are used. The results also show that even a small selection of S-C scheduler implementations demonstrates a wide range of different patterns of behavior. It is therefore suggested that selection of the most appropriate scheduler will largely depend on requirements of the application for which the scheduler is intended.

Keywords: scheduler, architecture, Master, Slave, jitter, time-triggered, co-operative, Shared-Clock, Controller Area Network, message latency

Cite This Article: Mouaaz Nahas, “Estimating Message Latencies in Time-Triggered Shared-Clock Scheduling Protocols Built on CAN Network.” *Journal of Embedded Systems* 2, no. 1 (2014): 1-10. doi: 10.12691/jes-2-1-1.

1. Introduction

Over recent years, researchers have looked at various ways in which time-triggered software architectures can be employed in low-cost embedded systems where reliability is a key design requirement [1-6]. Previous work in this area has considered the development of both single- and multi-processor designs. In the case of multi-processor embedded systems, it has been shown that a “Shared-Clock” (S-C) communication architecture – used along with “Time-Triggered Cooperative” (TTC) scheduling algorithm [7,8] – provides a simple and low-cost software framework for time-triggered operation when implemented on standard network protocols [1]. In such distributed systems, the Controller Area Network (CAN) protocol [9] is popular and cost-effective hardware communication solution [10,11,12,13].

Although originally designed for automotive applications, CAN has also been used in process control and many other industrial areas [10]. As a consequence of its popularity and widespread use, most modern microcontroller families have members with on-chip

hardware support for this protocol (e.g. [14,15,16,17]). Compared to modern protocols, CAN protocol can still be an appropriate solution for many systems due to its profound roots in automotive industry as well as its simplicity, low implementation costs and widespread availability [18]. Moreover, experience gained with CAN over the past years allows the creation of extremely reliable systems using this protocol [19]. CAN is usually viewed as “event-triggered” protocol [20], however, the use of a S-C architecture in conjunction with CAN hardware helps to achieve a time-triggered network operation [1].

The original S-C scheduling protocols were introduced by Michael Pont in 2001 [1]. In a more recent study [21], a set of four possible implementations of the S-C protocol including those presented in [1] were compared and documented. In [22], an alternative S-C scheduler was developed and compared with the four previously developed schedulers against a number of parameters including jitter behavior and resource requirements for practical implementations in low-cost embedded systems. The study demonstrates how the particular scheduling protocol – referred to as TTC-SCC5 – provided a valuable addition to the range of S-C schedulers developed for

reliable time-triggered embedded systems. However, detailed modeling and comparison of, for example, message latencies between communicating nodes in all schedulers have not been carried out. In [23,24], we developed a range of data coding techniques to reduce timing jitter caused by the CAN hardware bit-stuffing mechanism [10] and, hence, to improve the temporal behavior of the TTC-SCC1 scheduler. The present study attempts to assess the communication delays (latencies) between any two communicating nodes in the five previously developed S-C schedulers by providing mathematical equations. In particular, the equations are intended to estimate the message delays between the network Master and any Slave, any Slave and the network Master, and any two Slaves communicating to each other. As in [22], the five scheduling protocols will be referred to in this paper as “TTC-SCC1” - “TTC-SCC5” schedulers.

The remainder of this paper is organized as follows. Section 2 provides a general description of the TTC-SCC scheduling protocol considered in this study. In Section 3, the methodology used to estimate the time delays between communicating nodes is presented. In Section 4 – Section 8, we provide an overview of the TTC-SCC1 – TTC-SCC5 schedulers and present the message latency equations in the five schedulers (respectively). Section 9 provides a case study in which we compare the outlined schedulers in terms of their communication behavior. Finally, the overall paper conclusion is drawn in Section 10.

2. TTC-SCC Scheduler

In TTC-SCC systems considered here, the execution of tasks on the individual nodes are synchronized by sharing a single clock source between the various processor boards in the system. More clearly, on the Master node of a TTC-SCC network, a conventional time-triggered cooperative scheduler operates where the system is driven by periodic interrupts generated from an on-chip timer [1]. On the Slave nodes, a very similar scheduler operates. However, on the Slaves, no timer is used: instead, the Slave scheduler is driven by interrupts generated through the arrival of periodic “Tick” messages sent from the Master node. This means that all nodes are synchronized according to the Master clock (see Figure 1). Note that Master Tick message usually holds data for a particular Slave or a group of Slaves. Only the addressed Slave(s) must reply a form of acknowledgement “Ack” message to the Master straight after the Tick message has been received (for further details, see [1,22]).

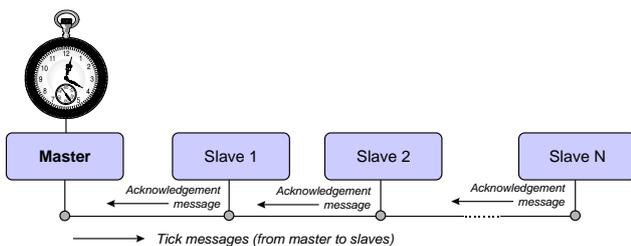


Figure 1. Simple architecture of shared-clock (S-C) scheduler

3. Methodology

Throughout this study, the transmission delays are calculated between the time at which an activity takes place in one node and the response to this activity in another node. This means that precise results can be obtained if delays are calculated between the time when data is generated in the sending node and the time when the receiving node begins to handle this data. If the activity on the sending node takes place at arbitrary instances, we evaluate the best-case (minimum) and the worst-case (maximum) message transmission times between the sending and receiving nodes.

In order to simplify the calculations, it has been assumed – throughout this study – that all tasks in the Master and Slaves have reasonably short execution times, thus the data is always generated close to the start of tick interval. Moreover, it has been assumed that the scheduler overhead time on Master and Slaves are very small and can hence be neglected. Based on these assumptions, the message latencies will be calculated between the start of the tick in which data is generated and the tick in which data is received.

4. TTC-SCC1 Scheduling Protocol

In this section, we review the TTC-SCC1 scheduler and derive the equations for message latencies between any two communicating nodes in the TTC-SCC1 network.

4.1. Overview

TTC-SCC1 is a simple version of the S-C scheduler. It employs a simple TDMA protocol in which the Master talks to one Slave only in each tick interval. The TDMA round is therefore proportional to the number of Slaves connected in the system (Figure 2). The two major concerns about this implementation are the possibility of having long TDMA round and the lack of support for Slave-to-Slave communication [21]. One consequence of having long TDMA round in this scheduler (where the Master talks to each Slave only once in the TDMA) is that a node failure may not be detected as quickly as required: such behavior can have a significant impact on system predictability. Complete description of this scheduler is detailed in [22]. Given that N is the number of Slaves and T is the tick interval, the TDMA round can be calculated as follows:

$$TDMA1 = NT \quad (1)$$

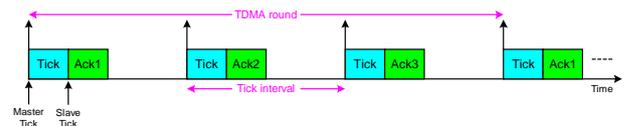


Figure 2. TDMA round for a four-node system using TTC-SCC1 scheduler

Given that M is the Master Tick message length, T is the tick interval, $TDMA1$ is the Time Division Multiple Access round and N is the number of Slaves, the message latencies between any two nodes in the network are calculated as follows.

4.2. Master-to-Slave Message Latency

In the best-case scenario, the data to be transmitted from the Master to a Slave at a given tick must be ready at the start of the tick interval and, therefore, it has already been generated in the task(s) executed within the preceding tick interval. The Master will hence be able to send this data with the Tick message due to transmit in the current tick. In contrast, in the worst-case scenario, the Master decides to send data to a given Slave straight after it has sent Tick message to that Slave.

Figure 3 shows an example where the Master node wants to communicate with S2 (i.e. Slave number 2). The best-case transmission process is illustrated using the “blue” color while the worst-case process is illustrated using the “red” color.

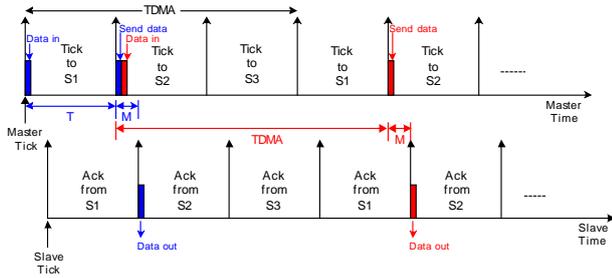


Figure 3. Master-to-Slave message latency in TTC-SCC1

The figure clearly shows that in the best-case scenario, the data – generated in the first tick – can be sent to S2 at the beginning of the second tick (where this tick is allocated to exchange data with S2). S2 can then extract the data on arrival of the Master Tick message. In the worst-case scenario, the Master needs to wait until the tick allocated for S2 – in the next TDMA round – arrives during which it can send a Tick message with data allocated for S2. Remember that Slave ticks are always delayed by M , since Slave scheduler is triggered by the arrival of the Master Tick message.

The equations for the best- and the worst-case message latencies between the Master and a given Slave are presented in the following table. Note that these equations are simply derived from the graphical representation illustrated in Figure 3.

Table 1. Master-to-Slave latency equations in TTC-SCC1

	Best-case latency	Worst-case latency
Master-to-Slave latency	$T + M$	$TDMA1 + M$

Please note that throughout this study, we assume that the scheduler overheads on Master and Slaves are very small and therefore their durations do not appear in the equations.

4.3. Slave-to-Master Message Latency

Based on the same explanation provided in Section 4.2, the best- and the worst-case message transmissions can be derived using Figure 4. Note that, in this case, although S2 replies its Ack message to the Master in the second tick interval (straight after receiving the Tick message), the Master will not check the contents of S2 Ack message until the start of the next tick (just before sending data to S3).

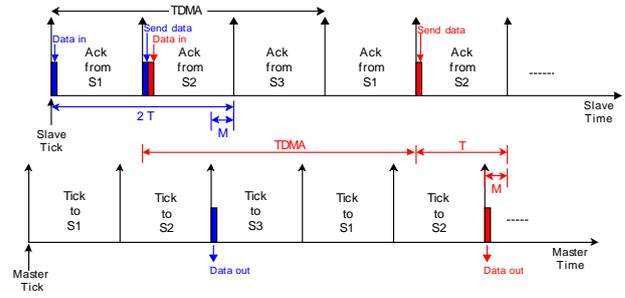


Figure 4. Slave-to-Master message latency in TTC-SCC1

Using the graphical representation illustrated in Figure 4, the equations for the best- and the worst-case message latencies between a given Slave and the Master are presented in the following table.

Table 2. Slave-to-Master latency equations in TTC-SCC1

	Best-case latency	Worst-case latency
Master-to-Slave latency	$2T - M$	$TDMA1 + T - M$

4.4. Slave-to-Slave Message Latency

In the Slave-to-Slave communication, the situation is more complicated. To be able to work out the message latency between any two Slaves in the TTC-SCC1 network, the shortest distance between their corresponding tick intervals (i.e. the tick intervals in which Slaves can send their “Ack” messages) must be calculated.

Given that X is the transmitting Slave and Y is the receiving Slave, the distance D_{XY} between “Ack- X ” and “Ack- Y ” is calculated as follows:

$$D_{XY} = [(Y - X) \bmod (N)]T \quad (2)$$

Where “mod” is “modulo arithmetic” function. For example, consider the example shown in Figure 2. The distance between Ack-1 and Ack-3, where $X = 1$ and $Y = 3$, is calculated as: $((3-1) \bmod (3))T = (2 \bmod (3))T = 2T$.

In contrast, the distance between Ack-3 and Ack-1, where $X = 3$ and $Y = 1$, is calculated as: $((1-3) \bmod (3))T = (-2 \bmod (3))T = T$. Note that the message latency between any two communicating Slaves is calculated as a function of D_{XY} . This is further illustrated in Figure 5 below.

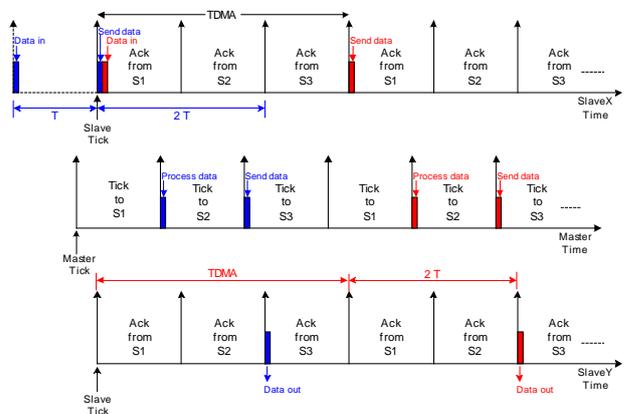


Figure 5. Slave-to-Slave message latency in TTC-SCC1

The figure illustrates the communication process between S1 and S3 in the TTC-SCC1 scheduler. In the

best-case scenario (blue color), S1 sends the data (which was generated in the preceding tick interval) with its Ack-1 message straight after the Master Tick message is received. The Master will check the contents of Ack-1 message in the following tick before it sends a Tick message addressing S2. The data can then be completely processed and placed in the corresponding CAN data registers for transmission with the following Tick message intended for S3. The diagram shows that this process takes time equals to $2T$ (the distance between Ack-1 and Ack-3) plus one additional tick interval.

In the worst-case scenario (red color), the data is generated in S1 after it has already sent its Ack-1 to the Master. This means that S1 can only send its data after a full TDMA round. This results in increasing the message latency between S1 and S3 to be equal to $TDMA1$ plus the distance between Ack-1 and Ack-3. Note that the process shown in Figure 5 presents the communication between any two Slaves when D_{XY} is larger than T (i.e. Ack messages for the communicating Slaves are not transmitted in consecutive tick intervals). When D_{XY} is equal to T , then the communication process in the described TTC-SCC1 becomes more complicated. This is simply because when the Master receives data from S1 – as an example – it cannot send it immediately to S2 since the data intended for S2 has already been configured and placed in the CAN data registers. This means that the Master always needs to wait for an extra TDMA round by which it can complete processing the data received from S1 and configure the Tick data message.

The equations for the best- and the worst-case message latencies between the Master and a give Slave are presented in the following table.

Table 3. Slave-to-Slave latency equations in TTC-SCC1

		Best-case latency	Worst-case latency
Slave-to-Slave latency	$D_{XY} > T$	$D_{XY} + T$	$D_{XY} + TDMA1$
	$D_{XY} = T$	$2T + TDMA1$	$T + 2 TDMA1$

Remember that in TTC-SCC1, $TDMA1 = NT$. By substituting this value in the equations shown, the results can be simplified as follows:

Table 4. Slave-to-Slave latency equations in TTC-SCC1 (simplified formula)

		Best-case latency	Worst-case latency
Slave-to-Slave latency	$D_{XY} > T$	$D_{XY} + T$	$D_{XY} + NT$
	$D_{XY} = T$	$(N+2)T$	$(2N+1)T$

Note that when the number of Slaves N significantly increases, the message latencies between the communicating Slaves will also increase by significant factors (except in the best-case scenario when $D_{XY} > T$). This implies that the described TTC-SCC1 may not be the appropriate solution for multi-processor designs with a large number of Slave nodes connected up in the network.

5. TTC-SCC2 Scheduling Protocol

In this section, we review the TTC-SCC2 scheduler and derive the equations for message latencies between any two communicating nodes in the TTC-SCC2 network.

5.1. Overview

The TTC-SCC2 scheduler was developed based on the TTC-SCC1 and intended to offer high flexibility in communication between the nodes. Initially, a simple example is given in which the Master can talk to one particular Slave every other tick interval. Such a Slave is described as a critical Slave which required to be checked by the network-Master regularly at high rates (Figure 6). For N Slaves and tick interval T , the TDMA round can be calculated as follows:

$$TDMA2 = (2N - 2)T \quad (3)$$

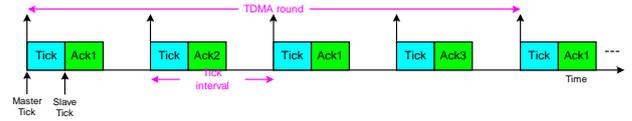


Figure 6. A simple TDMA configuration for a four-node system using TTC-SCC2 scheduler

A slightly more complicated example is provided in Figure 7 where it is assumed that the Master talks to the critical Slave at any frequency. Given that N is the number of Slaves, T is the tick interval, S_F is the frequently checked Slave, and F is the frequency of S_F ‘‘Ack’’ messages (in ‘‘ticks’’), the TDMA round can be calculated as follows:

$$\begin{aligned} TDMA2 &= (N - 1 + \frac{N - 1}{F - 1})T \\ &= (N + \frac{N - F}{F - 1})T = N.T + (\frac{N - F}{F - 1})T \end{aligned} \quad (4)$$

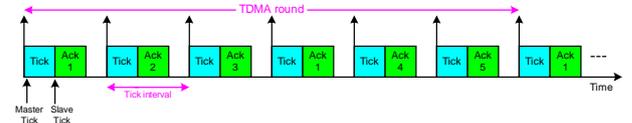


Figure 7. A more complicated TDMA configuration for a six-node system using TTC-SCC2 scheduler

It should be emphasized that the two examples represented only limited use of this scheduler. A more general example is therefore given in Figure 8 to show how the TDMA round can have a random pattern. This is entirely based on the application requirements.

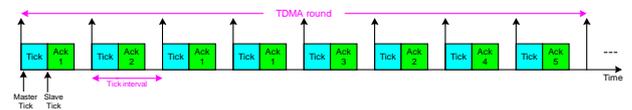


Figure 8. A TDMA configuration for a six-node system with arbitrary pattern using TTC-SCC2 scheduler

As with TTC-SCC1, this scheduler suffers lack of support for direct communication between Slaves, thereby causing the transmission time between any two Slaves to be comparatively long. Also the failure detection time of particular Slaves (which are checked less frequently) can be very long. Complete description of this scheduler is detailed in [22].

Calculating message latencies in TTC-SCC2 scheduler is not straightforward. This is because the Master communicates with Slaves in a random way depending on

the specification of the system for which the scheduler is used.

To get on with the calculations, it is important to define two parameters:

1. The distance between successive ticks allocated for a given Slave: this is referred to as D_{XX} .
2. The shortest distance between Ack messages from any two communicating Slaves: this is referred to as D_{XY} (as in TTC-SCC1).

Given that M is the Master Tick message length, T is the tick interval and $TDMA2$ is the Time Division Multiple Access round, the message latencies between any two nodes in the network are calculated as follows.

5.2. Master-to-Slave Message Latency

In the best-case scenario, the behavior is exactly same as that observed with TTC-SCC1 scheduler. However, in the worst-case scenario, after data is generated in a given tick, the Master needs to wait until the following tick in which it can communicate with the target Slave. This delay does not have to be as long as the TDMA round: instead, it depends on D_{XX} . The value of D_{XX} must lie between T and $TDMA2$. For example, if the Master communicates with the Slave only once in the TDMA round, D_{XX} will be equal to $TDMA2$. In contrast, if the Slave is allocated adjacent tick intervals to transmit its Ack message, then D_{XX} will be equal to T .

Figure 9 illustrates the process of Master to Slave 2 communication in the system shown in Figure 8. In the worst-case scenario, data – which is generated immediately after the Master sent Tick to S2 – can only be sent to S2 in the next tick allocated for this Slave. In the example shown, this delay is equal to $4T$. For S3, where only one tick in the whole TDMA round is allocated, D_{XX} will be equal to $TDMA2$, and so on.

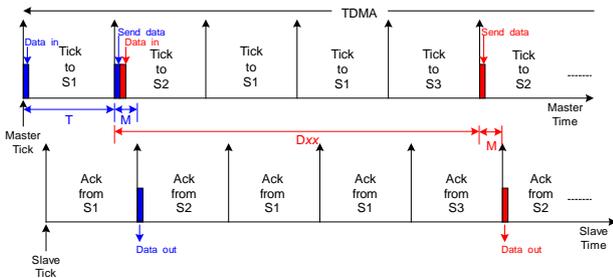


Figure 9. Master-to-Slave message latency in TTC-SCC1

A summary of the results is provided in the following table.

Table 5. Master-to-Slave latency equations in TTC-SCC2

	Best-case latency	Worst-case latency
Master-to-Slave latency	$T + M$	$D_{XX} + M$

Based on the discussion above, the worst-case Master-to-Slave latency will have the minimum value of $T + M$ and the maximum value of $TDMA2 + M$.

5.3. Slave-to-Master Message Latency

Again, the behavior here is similar to that observed in the TTC-SCC1 scheduler. The only difference – as in Master-to-Slave communication – is that the $TDMA2$ term is replaced by D_{XX} in the equations. A summary of the

results is provided in the following table. Remember that D_{XX} can have a value between T and $TDMA2$.

Table 6. Slave-to-Master latency equations in TTC-SCC2

	Best-case latency	Worst-case latency
Slave-to-Master latency	$2T - M$	$D_{XX} + T - M$

Similarly, the worst-case Slave-to-Master latency will have the minimum value of $2T - M$ and the maximum value of $TDMA2 + T - M$.

5.4. Slave-to-Slave Message Latency

The situation here is slightly more complicated. Since the communication between nodes in this scheduler has a random pattern, D_{XY} cannot be calculated as a function of X and Y (as with TTC-SCC1 scheduler). For example, the distance between the Slave 1 and Slave 3 cannot be calculated as $(3-1)T$.

In order to present a general formula for Slave-to-Slave message latency, it is important to know the “current” and the “next” distance between the Ack message of the sending Slave and the Ack message of the receiving Slave. The “current” distance is denoted by $D_{Xi Yi}$, while the “next” distance is denoted by $D_{X(i+1) Y(i+1)}$. For example, consider the communication between S1 and S2 in the system shown in Figure 8. For these two Slaves, $D_{Xi Yi} = T$ and $D_{X(i+1) Y(i+1)} = 3T$. In the same way, considering S1 and S3, $D_{Xi Yi} = T$ and $D_{X(i+1) Y(i+1)} = 4T$. Note that the “current” distance must be the shortest distance between the two communication Slaves and the “next” distance is the one follows it.

Accordingly, the message latencies between any two Slaves depend of both the “current” and “next” distances. In the best-case scenario, when data is generated in the previous tick to the current one, the message latency between S1 and S2 can be calculated as follows.

Table 7. Slave-to-Slave latency equations in TTC-SCC2 (best-case scenario)

	Best-case scenario	
Slave-to-Slave latency	$D_{Xi Yi} > T$	$D_{Xi Yi} + T$
	$D_{Xi Yi} = T$	$D_{Xi Y(i+1)} + T$

Please note that $D_{Xi Yi}$ denotes the distance between the Ack message of the sender and the consecutive Ack message of the receiver, while $D_{Xi Y(i+1)}$ is the distance between the Ack message of the sender and the one after the next Ack message of the receiver.

Likewise, in the worst-case scenario, when data is generated in the current tick after Ack message is sent, the message latency between S1 and S2 can be calculated as follows.

Table 8. Slave-to-Slave latency equations in TTC-SCC2 (worst-case scenario)

	Worst-case scenario	
Slave-to-Slave latency	$D_{X(i+1) Y(i+1)} > T$	$D_{Xi Y(i+1)}$
	$D_{X(i+1) Y(i+1)} = T$	$D_{Xi Y(i+2)}$

Please note that the best-case scenario here does not mean the shortest message latency, and the worst-case scenario does not mean longest message latency.

The figure shows the message latency between S1 and S2 in the example provided in Figure 8, where $D_{Xi Yi} = T$ and $D_{X(i+1) Y(i+1)} > T$. The figure shows that the best-case scenario produced longer message latency than in the worst-case scenario.

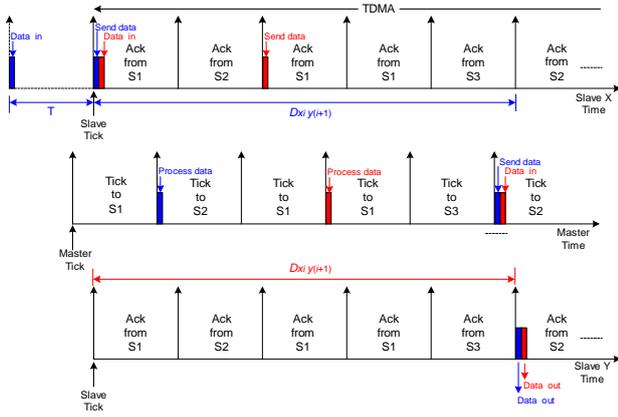


Figure 10. Slave-to-Slave message latency in TTC-SCC2

6. TTC-SCC3 Scheduling Protocol

In this section, we review the TTC-SCC3 scheduler and derive the equations for message latencies between any two communicating nodes in the TTC-SCC3 network.

6.1. Overview

The limitations observed in TTC-SCC1 and TTC-SCC2 schedulers were addressed through the development of a third implementation of the S-C scheduler which was called TTC-SCC3. Such an implementation allowed each Slave to send its messages to all Slaves in the network. It also allowed the Master to talk to all (or a group of) Slaves within a single tick interval, causing a significant reduction in the length of the TDMA round (Figure 11). Given that N is the total number of Slaves, m is the maximum number of Slaves replying per tick and T is the tick interval, the TDMA round can be calculated as follows:

$$TDMA3 = \frac{NT}{m} \quad (5)$$

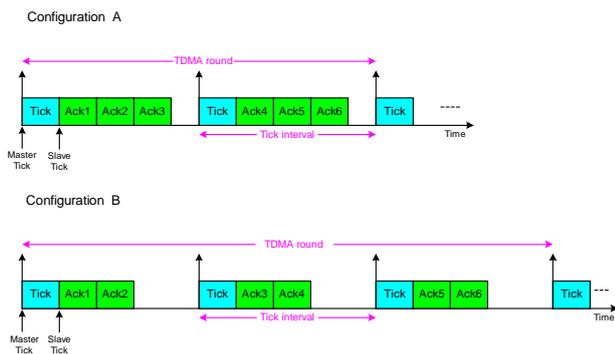


Figure 11. Two possible TDMA configurations for a seven-node system using TTC-SCC3 scheduler

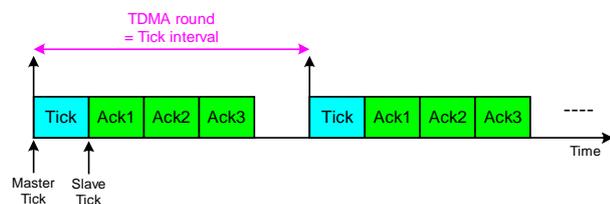


Figure 12. A simple TDMA configuration for a four-node system using TTC-SCC3 scheduler

If all Slaves are permitted to transmit in the same tick interval, the TDMA round will be equal to the tick interval T (see Figure 12).

The major problem in this and all the previous schedulers is the high levels of jitter at the release time of Slave tasks due to variations in the length of Tick messages. Such variations are dependent on the nature of the transmitted data in the Tick messages [22]. If (for example) data sent in the Tick messages are likely to be random, this can have more impact on the timing of Slave tasks. Complete description of this scheduler is detailed in [24].

Given that M is the Master Tick message length, T is the tick interval, $TDMA3$ is the Time Division Multiple Access round, N is the number of Slaves, and m is the maximum number of Slaves replying per tick interval, the message latencies between any two nodes in the network are calculated as follows.

6.2. Master-to-Slave Message Latency

Basically, the results obtained here are similar to those obtained from the TTC-SCC1 and TTC-SCC2. However, since the TDMA round is shorter in the TTC-SCC3, this results in a reduced message latency. A summary of the results is provided in the following table.

Table 9. Master-to-Slave latency equations in TTC-SCC3

	Best-case latency	Worst-case latency
Master-to-Slave latency	$T + M$	$TDMA3 + M$

Please note that in the example given in Figure 12 (where $TDMA3 = T$), the Master-to-Slave latency is fixed and always equal to $T + M$.

6.3. Slave-to-Master Message Latency

Again, the equations for the Slave-to-Master message latencies are similar to those derived before (in TTC-SCC1 and TTC-SCC2). But, again, the message latencies are expected to be much shorter in the TTC-SCC3 due to a reduced TDMA round. A summary of the results is provided in the following table.

Table 10. Slave-to-Master latency equations in TTC-SCC3

	Best-case latency	Worst-case latency
Slave-to-Master latency	$2T - M$	$TDMA3 + T - M$

Also note here that in the example given in Figure 12 (where $TDMA3 = T$), the Slave-to-Master latency is fixed and always equal to $2T - M$.

6.4. Slave-to-Slave Message Latency

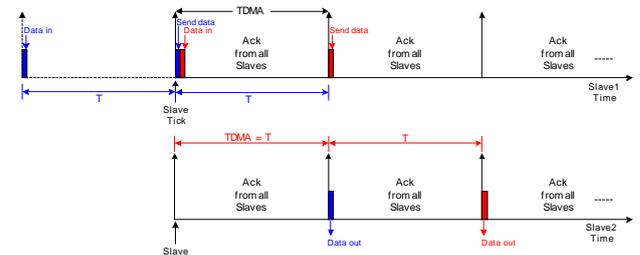


Figure 13. Slave-to-Slave message latency in TTC-SCC3

Here, there is a substantial difference between the behavior of TTC-SCC3 scheduler and the previous schedulers. Since all Slaves are configured to receive Ack messages sent from other Slaves, Slave to Slave message latency is substantially reduced. This is further illustrated in Figure 13.

Assume that S1 wants to send data to S2. In the TTC-SCC3 scheduler described, Slave-to-Slave communication can be made directly without going through the Master. More clearly, in the best-case scenario, data on S1 must be ready to transmit at the start of the tick (i.e. data has been generated in the previous tick interval). The data will then be sent out in the Ack-1 message to all nodes. At the beginning of the following tick (for which the data is intended) will check the contents of Ack-1 message and hence extract the requested data for use in that tick. In the worst-case scenario, where S1 decides to send the data straight after transmitting Ack-1 message, it has to wait for a full TDMA round (which is equal to T in the simple implementation shown in Figure 13) before which it can send the data out with the next Ack-1 message to all Slaves. Once S2 receives the Ack-1 message, the scheduler on S2 needs only one tick to process the Ack-1 message and hence extract the requested data.

Please note that the latencies between Slaves are almost same as the latencies between a given Slave and the Master. These results are summarized in the following table.

Table 11. Slave-to-Slave latency equations in TTC-SCC3

	Best-case latency	Worst-case latency
Slave-to-Slave latency	$2T$	$TDMA3 + T$

Again note that in the example given in Figure 13 (where $TDMA3 = T$), the Slave-to-Slave latency is fixed and always equal to $2T$.

Please note that a large value of m (the number of Slaves replying per tick) would require that the tick interval should be extended to accommodate m Ack messages sent from m Slaves. This increase in the scheduler tick interval may not be appropriate for some applications where tick interval has to be extremely short. This means that it is always a trade-off between message latencies and tick interval.

7. TTC-SCC4 Scheduling Protocol

In this section, we review the TTC-SCC4 scheduler and derive the equations for message latencies between any two communicating nodes in the TTC-SCC4 network.

7.1. Overview

The TTC-SCC4 scheduler provided an attractive solution to the jitter problem caused by variations in the transmission time of Tick messages. This was achieved by allocating a separate node for generating the heartbeat of the network. This node was seen as a “tick-only-Master” node which processes no data. The Master node in previous implementations becomes an ordinary Slave node which only sends data messages (Figure 14). Apart from this feature, the same message configuration – as in TTC-SCC3 – was used with this scheduler. Given that N is the total number of Slaves, m is the maximum number of Slaves replying per tick and T is the tick interval, the TDMA round can be calculated as follows:

$$TDMA4 = \frac{(N+1)T}{m} \quad (6)$$

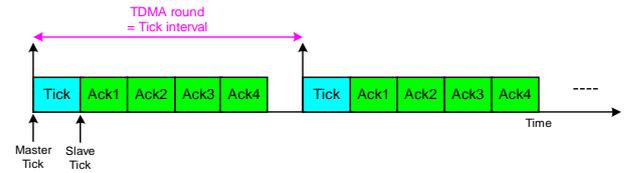


Figure 14. A simple TDMA configuration for a four-node system using TTC-SCC4 scheduler

The only problem with TTC-SCC4 scheduler is that it required an additional microcontroller board only to send Tick messages while not being involved in any other activities. This obviously caused a reduction in the resource efficiency. Complete description of this scheduler is detailed in [22].

Overall, since the TTC-SCC4 scheduler was developed only to deal with jitter problem in the TTC-SCC3 scheduler, the message latencies obtained from the TTC-SCC4 scheduler are expected to be similar to those presented for the TTC-SCC3 scheduler.

However, in the TTC-SCC3 scheduler, it is assumed that Master and Slave should exchange “real” data between them, therefore the equations for Master-to-Slave and Slave-to-Master derived for the TTC-SCC3 do not work here anymore. This is again because the Master node in the TTC-SCC4 scheduler cannot send data to (or respond to data from) Slave nodes. In order to assess the TTC-SCC4 scheduler behavior (in the same way as with the previous schedulers), it must be assumed here that the additional Slave node will completely replace the original Master node in processing data, but will not be superior to other Slaves. Therefore, Master-to-Slave and Slave-to-Master message latencies will be identical to Slave-to-Slave message latencies (in the context discussed here).

7.2. Master-to-Slave Message Latency

Same as Slave-to-Slave message latency in TTC-SCC3 scheduler (See Section 6.4).

7.3. Slave-to-Master Message Latency

Same as Slave-to-Slave message latency in TTC-SCC3 scheduler (See Section 6.4).

7.4. Slave-to-Slave Message Latency

Same as Slave-to-Slave message latency in TTC-SCC3 scheduler (See Section 6.4).

8. TTC-SCC5 Scheduling Protocol

In this section, we review the TTC-SCC5 scheduler and derive the equations for message latencies between any two communicating nodes in the TTC-SCC5 network.

8.1. Overview

In order to combine the features of TTC-SCC3 and TTC-SCC4 schedulers without adding more cost to the system, the TTC-SCC5 scheduler was developed. Simply, such a scheduler allowed the Master node to send two

types of messages consecutively. The first message was only to trigger the Slave nodes at precisely-fixed intervals, where the following message was designated for Master data intended for all or some Slaves. Particularly, the Master is configured to send out two types of messages: Tick messages and Data messages. As with the TTC-SCC4 scheduler, the Tick messages are configured to have “empty” data. This, again, means that these messages are only used to generate the time-reference for the whole network while processing no data. After a Tick message is sent out to all Slaves at each tick, the Master can then send its data in its Data message (see Figure 15). The TDMA round in TTC-SCC5 scheduler is calculated in the same way as in TTC-SCC3 scheduler (i.e. $TDMA5 = TDMA3$).

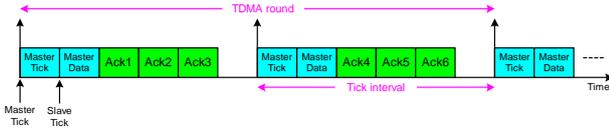


Figure 15. A TDMA configuration for a seven-node system using TTC-SCC5 scheduler

Although the bandwidth utilisation might be slightly reduced, due to the scheduling of additional messages in the tick intervals, TTC-SCC5 can provide a highly-predictable message and task operations compared to all previous implementations. Complete description of this scheduler is detailed in [22].

Given that M_T is the Master Tick message length, M_D is the Master Data message length, T is the tick interval, $TDMA5$ is the Time Division Multiple Access round, N is the number of Slaves, and m is the maximum number of Slaves replying per tick interval, the message latencies between any two nodes in the network are calculated as follows.

8.2. Master-to-Slave Message Latency

Using TTC-SCC5 protocol, the Master-to-Slave communication process will slightly be different than that achieved with the TTC-SCC3 and TTC-SCC4. This process is described here.

By considering the best-case scenario, the data sent from the Master to Slaves is assumed to be generated in the previous tick. However, this data will be sent with the Master Data message (not with the Tick message). The recipient Slaves will, therefore, process the received data in the tick following the tick in which the Master Data message is received (in the same way the Ack messages are treated by the Master and by other Slaves: see Figure 13). As a result, the best-case transmission time between the Master and any Slave will be one tick longer than that achieved with the previous S-C protocols.

Since the Master node is allowed to transmit its Data message every tick, it does not have to wait for a full TDMA round before it sends its data in the worst-case scenario: instead, it is able to send its data in the tick

following the tick in which the data is generated. Therefore, the best- and the worst-case transmission latencies will always be identical in this scheduler. Please remember that the Slave clocks are always delayed – according to the Master clock – by the value of M_T which represents the length of the Master “empty” Tick message. This is why this term appears in the equations. A summary of the results is provided in the following table.

Table 12. Master-to-Slave latency equations in TTC-SCC5

	Best-case latency	Worst-case latency
Master-to-Slave latency	$2T + M_T$	$2T + M_T$

The results, in the table, show that the Master-to-Slave message latency in this scheduler is always fixed and equal to $2T + M_T$.

8.3. Slave-to-Master Message Latency

The Slave-to-Master communication process is identical to that achieved when the TTC-SCC3 or TTC-SCC4 is used. Again, remember that the Slave clocks are always delayed by the value of M_T . A summary of the results is provided in the following table.

Table 13. Slave-to-Master latency equations in TTC-SCC5

	Best-case latency	Worst-case latency
Slave-to-Master latency	$2T - M_T$	$TDMA5 + T - M_T$

8.4. Slave-to-Slave Message Latency

The latencies between the Slaves in this scheduler are similar to those obtained from the TTC-SCC3 and the TTC-SCC4. A summary of the results is provided in the following table.

Table 14. Slave-to-Slave latency equations in TTC-SCC5

	Best-case latency	Worst-case latency
Slave-to-Slave latency	$2T$	$TDMA5 + T$

9. Case Study: Practical Comparison Between TTC-SCC Schedulers

To provide a practical comparison between the communication behavior in the various schedulers considered, a small case study is used. The case study is based on the system with the following specifications. In this system, three Slave nodes are connected up in the network, CAN baudrate is 1 Mbit/s and the tick interval is 4 ms. Assuming “standard” CAN messages (i.e. 11-bit identifier), “Tick” and “Ack” messages send seven “random” data bytes along with the Slave / Group ID byte (except in the Tick-only message which has no data), then the value of M , M_D and S are equal to 135 μ s (with the worst-case level of bit-stuffing) and the value for M_T is equal to 47 μ s (without data bytes and any bit-stuffing) [9]. The TTC-SCC schedulers used with this small network has the following configurations:

Table 15. TTC-SCC models used in the case study to allow a comparison between schedulers

Scheduler name	Model	TDMA (μ s)	Comments
TTC-SCC1	Figure 2	12	TDMA round consists of three ticks.
TTC-SCC2	Figure 6	16	TDMA round consists of four ticks. S1 is allocated two ticks to send its Ack message, while S2 and S3 only send their Ack once.
TTC-SCC3	Figure 12	4	TDMA = T , $m = 3$. All Slaves send their Ack in the same tick
TTC-SCC4	Figure 14	4	TDMA = T , $m = 4$. The number of Slaves is increased by one. Tick message is very short compared to Slaves Ack messages.
TTC-SCC5	Figure 15	4	TDMA = T , $m = 3$. Tick message is also very short compared to Master Data and Slaves Ack messages.

The results obtained from this case study are summarized in the following table. Note that the following abbreviations are used: M-S (Master-to-Slave), S-M (Slave-to-Master), S-S (Slave-to-Slave), BC (Best-case) and WC (Worst-case).

Table 16. Results from the case study used to compare between TTC-SCC schedulers

Scheduler name	M-S1 Latencies (μ s)		S1-M Latencies (μ s)		S1-S2 Latencies (μ s)	
	BC	WC	BC	WC	BC	WC
TTC-SCC1	4.135	12.135	7.865	15.865	20	28
TTC-SCC2	4.135	8.135	7.865	11.865	20	24
TTC-SCC3	4.135	4.135	7.865	7.865	8	8
TTC-SCC4	8	8	8	8	8	8
TTC-SCC5	8.047	8.047	7.953	7.953	8	8

The results in the table clearly show the difference between communication behavior of the five compared schedulers. In more details, it is clear the TTC-SCC1 – although very simple and efficient – can produce long delays in communication between nodes, especially when the worst-case scenario is considered. The TTC-SCC2 provides some improvement to these parameters. The simple case study used here evaluated the communication latencies between S1 (which is more frequently checked) and S2 which is checked once in the TDMA round. If both Slaves are checked only once and the TDMA round increases, the message latencies would be expected to increase correspondingly, with the result that TTC-SCC2 may not be a good alternative to TTC-SCC1 for some systems.

Moving on to the next schedulers, it is clear from the results that in TTC-SCC3, TTC-SCC4 and TTC-SCC5 – where all Slaves are permitted to transmit their Ack messages simultaneously – the message latencies have been reduced significantly.

Comparing TTC-SCC4 and TTC-SCC5, the results look almost the same. Remember that TTC-SCC5 was built on TTC-SCC4 and aimed to provide the same level of performance at lower cost. When comparing TTC-SCC5 with TTC-SCC3, Master-to-Slave latencies are shorter (almost by half) in the TTC-SCC3. Apart from those, the performance is similar. Remember that in TTC-SCC3, jitter levels are quite high as compared to those obtained from the TTC-SCC5.

10. Conclusions

A Shared-Clock (S-C) scheduler has been developed as a “high-level” protocol to provide “time-triggered” network operation for CAN-based and other hardware protocols which are viewed as “event-triggered”. Various ways in which time-triggered S-C scheduler can be implemented in low-cost embedded systems were presented in previous work. This paper aimed to assess and compare the wide range of previously developed S-C scheduling protocols in terms of their communication behavior. In particular, message latencies between any two communicating nodes in each scheduler were estimated using mathematical formulas. This process was intended to help embedded systems developers select the most appropriate S-C scheduler for their existing CAN-based implementations.

The results presented in the paper show that even a small (and by no means exhaustive) selection of S-C scheduler implementations demonstrated a wide range of different patterns of behavior. Overall, there is no perfect implementation which may fit all applications. The selection of a particular implementation will, hence, depend on the requirements of the application in which the scheduler is employed.

Acknowledgement

The work presented in this paper was carried out in the Embedded Systems Laboratory (ESL) at University of Leicester, UK, under the supervision of Professor Michael Pont, to whom the author is thankful.

References

- [1] Pont, M, Patterns for Time-Triggered Embedded Systems: Building Reliable Applications with the 8501 Family of Microcontrollers, Addison Wesley, 2001.
- [2] Pont, M, “An object-oriented approach to software development for embedded systems implemented using C,” *Transactions of the Institute of Measurement and Control*, 25 (3), 217-238. 2003.
- [3] Pont, M. J., and Banner, M. P, “Designing embedded systems using patterns: A case study,” *Journal of Systems and Software*, 71 (3), 201-213. 2004.
- [4] Kurian, S., and Pont, M. J, “The maintenance and evolution of resource-constrained embedded systems created using design patterns,” *Journal of Systems and Software*, 80 (1), 32-41. 2007.
- [5] Wang, H., Pont, M. J., and Kurian, S, “Patterns which help to avoid conflicts over shared resources in time-triggered embedded systems which employ a pre-emptive scheduler,” *SAE Transactions*, 115 (7), 795-83. 2005.
- [6] Nahas, M, “Employing Two “Sandwich Delay” Mechanisms to Enhance Predictability of Embedded Systems Which Use Time-Triggered Co-operative Architectures,” *International Journal of Software Engineering and Applications*, 4 (7), 417-425. 2011.
- [7] Baker, T. P, and Shaw, A., “The cyclic executive model and Ada,” *Real-Time Systems*, 1 (1), 7-25. 1989.
- [8] Locke, C. D, “Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives,” *Real-Time Systems*, 4 (1), 37-53. 1992.
- [9] Bosch, CAN Specification Version 2.0, Robert Bosch GmbH, 1991.
- [10] Farsi, M, and Barbosa, M. B., CANopen implementation: applications to industrial networks, Research Studies, 1999.
- [11] Fredriksson, L. B, “Controller Area Networks and the protocol CAN for machine control systems,” *Mechatronics*, 4 (2), 159-172. 1994.
- [12] Thomesse, J. P, “A review of the fieldbuses,” *Annual reviews in Control*, 22, 35-45. 1998.
- [13] Sevillano, L. J, Pascual, A., Jimenez, G., and Civit-Balcells, A., “Analysis of channel utilization for controller area networks,” *Computer Communications*, 21 (16), 1446-1451. 1998.
- [14] Philips, P8x592 8-bit microcontroller with on-chip CAN, Philips Semiconductor, 1996.
- [15] Siemens, C515C 8-bit CMOS microcontroller, user’s manual, Siemens, 1997.
- [16] Infineon, C167CR Derivatives 16-Bit Single-Chip Microcontroller, Infineon Technologies, 2000.
- [17] Philips, LPC2119/2129/2194/2292/2294 microcontrollers user manual, Philips Semiconductor, 2004.
- [18] Ayavoo, D, The development of reliable X-by-wire systems: assessing the effectiveness of a ‘simulation first’ approach, PhD Thesis, University of Leicester. 2006.
- [19] Short, M., and Pont, M. J, “Fault-tolerant time-triggered communication using CAN,” *IEEE Transactions on Industrial Informatics*, 3 (2), 131-142. 2007.
- [20] Leen, G., and Heffernan, D, “TTCAN: a new time-triggered controller area network,” *Microprocessors and Microsystems*, 26 (2), 77-94. 2002.

- [21] Ayavoo, D., Pont, M. J., Short, M., and Parker, S, "Two novel shared-clock scheduling algorithms for use with CAN-based distributed systems," *Microprocessors and Microsystems*, 31 (5). 326-334. 2007.
- [22] Nahas, M, "Developing a Novel Shared-Clock Scheduling Protocol for Highly-Predictable Distributed Real-Time Embedded Systems," *American Journal of Intelligent Systems*, 2 (5). 118-128. 2012.
- [23] Nahas, M., Pont, M. J., and Short, M, "Reducing message-length variations in resource-constrained embedded systems implemented using the Controller Area Network (CAN) protocol," *Journal of Systems Architecture*, 55 (5). 344-354. 2009.
- [24] Nahas, M, "Applying Eight-to-Eleven Modulation to reduce message-length variations in distributed embedded systems using the Controller Area Network (CAN) protocol," *Canadian Journal on Electrical and Electronics Engineering*, 2 (7). 282-293. 2011.