

# SCHEDULING FOR A TTCAN NETWORK WITH A STOCHASTIC OPTIMIZATION ALGORITHM

**Fernanda Coutinho**

**Jorge Barreiros**

Superior Institute of Engineering of Coimbra  
Quinta da Nora  
3030 Coimbra, Portugal

**José Alberto Fonseca**

Dep. Electronic and Telecommunications  
Of University of Aveiro  
Campus Santiago  
3800 Aveiro, Portugal

**Abstract:** TTCAN (*Time-Triggered Controller Area Network*) is a new session level time-triggered protocol defined for the CAN fieldbus. Before transmitting a message set over a TTCAN network, it is necessary to create a valid scheduling table that specifies how the time is discretized into time-slots and how the message instances are allocated into those time-slots. Building a valid optimised scheduling table that follows the TTCAN specification isn't trivial because several distinct optimization criterions should be considered. Therefore, we have developed an automated stochastic scheduling tool for the TTCAN protocol that will generate a valid scheduling table with low message jitter. The tool was applied to two well-known benchmarks from the automotive industry and succeeded in generating valid solutions for those benchmarks. Additionally, the jitter in those solutions was considerably reduced when an additional optimization phase was performed after scheduling. *Copyright © 2001 IFAC*

**Keywords:** Scheduling algorithms, Fieldbus, Jitter.

## 1. INTRODUCTION

Automotive industry has been using the CAN fieldbus for real-time distributed systems embedded in vehicles. An on-going standardisation process is defining time-triggered based communication to be used in such applications, in order to profit from the properties of this communication paradigm (Kopetz, 1997). The future standard will use a scheduling table that must be built off-line, prior to the system start of operation. In this paper, we present the first version of a tool that can be used to simplify the construction of the required scheduling table. We begin with a brief presentation of the TTCAN protocol, and then we follow by describing the scheduler and optimiser algorithms. We show the results obtained from applying the tool to the SAE and PSA benchmarks (Tindel et al, 1994; Navet et al, 1997). These results are briefly

analysed, some conclusions are extracted and some future work is identified.

## 2. TTCAN

During the last years, CAN – Controller Area Network has become the main fieldbus used in the time critical parts of automotive systems. Although CAN operates following the event-triggered paradigm, recent academic studies such as (Kopetz, 1997) pointed to the possibility of using a time-triggered approach in part of the operation of CAN based distributed systems. This feature could improve the bandwidth utilisation in CAN-based real-time systems by separating the periodic traffic from the aperiodic one and keeping the timeliness guarantees of the former. From the industrial side, the interest of using the time-triggered approach in CAN has also been recognised since some time and, as a consequence, a standardisation task force

(ISO/TC 22/SC 3/WG 1/TF 6) was launched under International Standards Organisation in order to achieve a definition for a session layer for CAN. This standard, which will be numbered 11898-4, is currently under final appreciation before balloting. It is already known as TTCAN, from Time-Triggered Controller Area Network.

In TTCAN, a special node, the time master, is responsible for the transmission of a systolic message, the reference message, which is used to achieve synchronisation between the fieldbus nodes. The reference message marks the start of a time slot which is called the Basic Cycle (BC). The BC may be divided in different types of windows, namely, the exclusive windows and the arbitrating windows. An exclusive window is used to transmit a specific periodic message. An arbitrating window may be used to transmit any message, provided it gains the bitwise arbitrating process as in normal CAN operation and provided it finishes transmission before the end of the window, thus guaranteeing temporal isolation between the different types of traffic.

The complete traffic pattern in a TTCAN system consists in a fixed number of consecutive BCs and is named the System Matrix or Matrix Cycle (MC). In figure 1, an example of a system MC is presented. There, it is possible to notice one of the major restrictions in the MC construction, which is the column like organisation. All the windows of the same column in every BC must be of the same size. With this organisation it is possible to define a set of trigger instants (offsets from the reference message) which is kept constant from BC to BC all along the MC, thus simplifying the TTCAN controllers hardware. Once the MC is defined, it is possible to merge two (or more) consecutive arbitrating windows in the same BC in order to facilitate the transmission of normal CAN messages. Finally, The other major restriction is in the number of BCs per MC which must be an integer power of two.

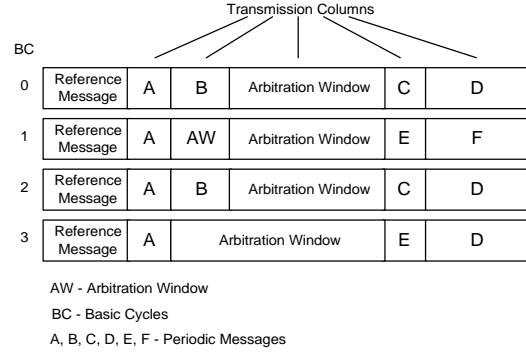


Figure 1 Example of a Matrix Cycle in TTCAN.

In TTCAN, the system matrix must be defined off-line. All the nodes must have stored the correspondent information prior to the start of operation. This leads to a complete inflexibility of TTCAN in what concerns transmitting periodic traffic. Details about TTCAN operation modes, synchronisation techniques and other specification details can be found in (Fuhrer et al, 2000; Hartwich et al, 2000) and in the draft standard.

### 3. SCHEDULER

Scheduling a message set for transmission over a TTCAN network consists in building the system matrix (SM). The SM is built by specifying the number of columns, the duration of each column, the number of rows and the time-slot assignment. These specifications should ensure that message period and transmission time are respected. It is usually possible to build several distinct SMs for the same message set. Consequently, it is possible to rank several distinct matrices according to some predefined quality criterion. In the experiments we conducted, we decided to rank matrices according to the message period jitter. This jitter is computed by averaging the difference between expected period and effective period over the macro cycle of the system for all messages (Equation 1).

$$Jitter = \sum_p \frac{\sum_i |e_i^p - a_i^p|}{M} \quad (1)$$

$e_i^p$  is the expected time for beginning of transmission of instance  $i$  of message  $p$  according to desired period,  $a_i^p$  is the actual time of beginning of transmission and  $M$  is the macro cycle length.

Our scheduler builds the system matrix in two steps:

- ✦ **Scheduling** – A feasible set of distinct SMs is generated.
- ✦ **Optimization** – The matrices are optimised and selected according to a given criterion, in this case, lower message period jitter.

The optimization is a stochastic process, so the resulting matrix and quality of the solution will not always be the same. The scheduling process, however, will always generate similar (but not identical, see below) SMs.

### 2.1. Scheduling Process

To ensure that the average periods of the messages are maintained, the SM should have an area of  $M$ , where  $M$  is the least common multiple of the message's periods. The desired average message period  $P_i$  for message  $i$  will then be obtained if there are exactly  $M/P_i$  instances of that message in the scheduling matrix.

The scheduling process is done in the following steps:

1. Determination of the maximum number of lines of the system matrix
2. Message allocation
3. Free time redistribution

There is an upper bound on the number of lines that can be used to generate a valid SM for a given message set. It is easily shown that the number of lines needs to be bounded at least by  $M/T_{max}$ , where  $T_{max}$  is the maximum transmission time of all the messages in the set (the duration of a matrix line is always equal or greater than  $T_{max}$ , so the resulting matrix area would be greater than  $M$  if more than  $M/T_{max}$  lines were used). In practice, however, the limit is much smaller (because the duration of a line usually needs to be much larger than  $T_{max}$ ). Because of this, the maximum line number is determined by the iterative increase of the number of lines until message allocation fails (see below). If the maximum number of lines is 0, then the set is considered not schedulable.

The following procedure is used for generating a feasible SM with a given number of lines  $L$ :

1. Messages are ordered by decreasing time of transmission into a ordered set  $I$ .
2. The first  $L$  instances are removed from the set. These instances will all be allocated to the same column in the system matrix. The duration of the column is set to be the greatest transmission time of the removed messages.
3. Repeat step 2 until  $I$  is empty.
4. Pad the remaining time with empty columns.

Messages are ordered by decreasing time of transmission in order for the algorithm to generate the SM with lowest sum of column duration ( $S$ ) possible. If the duration  $S$  after step 3 is greater than  $M/L$ , then it is impossible to generate a valid scheduling table with this number of lines (this is the test used to determine the maximum number of lines (see above)). When this is not the case, the remaining time  $(M/L)-S$  will be allocated to empty columns to fill up the space. For optimization purposes, it is convenient to have an empty column following every occupied column (even if its duration is 0), so we double the number of columns and distribute the remaining time randomly among those. These empty columns will separate the columns generated in steps 2 and 3. The duration of these columns is the only random factor in the scheduler output matrices.

Figure 2 shows the resulting scheduling matrix for a set of four messages, with periods of 10, 12, 15 and 20 and transmission time 1. Note that although no columns are visible within the first and second pairs of occupied columns, they do exist.

A	B		C	A		D	
A	B		C	A		D	
A	B		C	B			
A	B		C	D			

Figure 2 – Example of a system matrix after the scheduling process.

### 2.2. Optimization Process

The optimization process uses a set of feasible system matrices constructed as described above. These matrices will be changed by iteration, and the matrix that offers the lower jitter is considered to be the best when the optimization ends. These matrices are changed by applying one of several predefined transformations. These transformations generate a new system matrix from the original matrix, and they always ensure that the resulting matrix is also feasible.

The optimization process goes as follows:

1. Generate a set of system matrices. Each one of these matrices has a random number of lines (although always a power of 2, as defined in the TTCAN protocol), from 1 up to the maximum line number.
2. Run an initial transient phase, in order to increase the diversity of the matrices in the set.
3. Randomly select one of the matrices and apply a random transformation.
4. If this new matrix is better (has less jitter) than the worst matrix included in the set, then the later then one is replaced by the former.
5. Repeat from 3 until a maximum number of iterations is achieved.

After building the initial matrix set, a transient phase occurs where the transformations are applied randomly without any concern of improving the quality (reducing the jitter). The objective of this phase is to generate a greater diversity within the matrix set. This is necessary because all system matrices generated by the scheduler are very similar, and so, if no transient phase was used, we would begin the search always near the same point of the search space. There would be no advantage in using multiple alternative scheduling matrices, because they would all be the similar.

The optimization algorithm itself is based on a steady state genetic algorithm (Coutinho et al, 2000; Michalewicz, 1999), although it can hardly be considered a genetic algorithm ,essentially because of the lack of a crossover operator. We didn't find an effective way to maintain the feasibility of the generated solutions if a crossover operator was used.

The optimization is done by the consecutive application of one of several predefined matrix

transformations. If the resulting matrix is better than the worst matrix included in the set, then it replaces it. This is repeated until a maximum number of user-defined iterations is achieved.

In the end of the optimization, an additional step is taken to eliminate columns with duration 0 and to merge adjacent empty columns into a single column.

### 2.2.1 Matrix Transformations

The matrix transformations are applied with user-defined probabilities. In this version of the scheduler, we are using the following transformations:

- ✘ **Cell swap**- The contents of two random matrix cells are swapped. If message instances are transmitted in either one of the cells, then a check is made to ensure that the duration of the destination cell is sufficient to transmit that instance. If not, the transformation is aborted and the matrix is unchanged
- ✘ **Column swap** – Two randomly selected columns are swapped.
- ✘ **Free time redistribution** – A random percentage of the time slack in a random column (the time slack is the difference between the column duration and the longest transmission time of the instances transmitted in that column) is allocated to another column. One reason for considering columns even with duration 0 is because their duration can be increased in the future with this operator.
- ✘ **Vertical Mirror** – A random number of adjacent columns are selected and mirrored across the vertical axis of the selected columns.
- ✘ **Horizontal Mirror** – A random rectangular area is selected inside the matrix, and the contents of the cells in that area are mirrored across its horizontal axis.

Every one of this transformations always generates a new matrix that is also valid. Additional transformations are being considered and tested.

		A	C		B		D	
A		B	C				A	
D	B		A	C				B
A		D	C		B		A	

Figure 3 - Sample scheduling table after optimization.

#### 4. EXPERIMENTS

The experiments we had done used two well known benchmarks, coming from the automotive industry: the SAE benchmark and the PSA benchmark, with different transmission rates (125, 250 and 500 kbps) (Tindel et al, 1994; Navet et al, 1997). No results were obtained for the SAE message set at 125 Kbps, because it is not schedulable.

Some initial runs were made to determine the best settings for transformation probabilities for each benchmark, and then 20 optimising runs were made with those settings. The length of the optimization phase was set to 5000 iterations in the SAE benchmark and 10000 in the PSA benchmark. These values were selected to ensure a reasonably short experimentation time of a few minutes for each single run, but further improvements were observed by extending the number of iterations in any of the benchmarks.

For assessing the efficiency of the optimization phase, we used our scheduler to generate a set of 30 solutions without running the optimization phase (although the transient phase was executed, to ensure a diverse set of scheduling tables were generated). The following table shows the jitter of the best table in 20 runs, compared to the jitter of the non-optimised scheduling tables:

PSA125 Kbps	With Optimization	Without Optimization
Average Jitter	2,838248095	9,013082857

PSA250 Kbps	With Optimization	Without Optimization
Average Jitter	4,226845714	9,357181905

PSA500 Kbps	With Optimization	Without Optimization
Average Jitter	4,368542857	9,401934763

SAE250 Kbps	With Optimization	Without Optimization
Average Jitter	17,225212	28,74871

SAE500 Kbps	With Optimization	Without Optimization
Average Jitter	18,16949	29,581224

The following graphics show the jitter by message for each of the benchmarks and baudrates. As in the previous table, we present the values for the best table with Optimization and best table without Optimization.

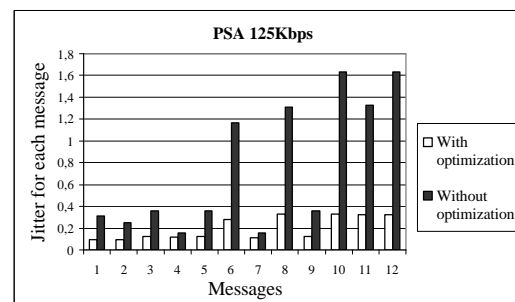


Figure 4 - Results for PSA benchmark at 125 kbps.

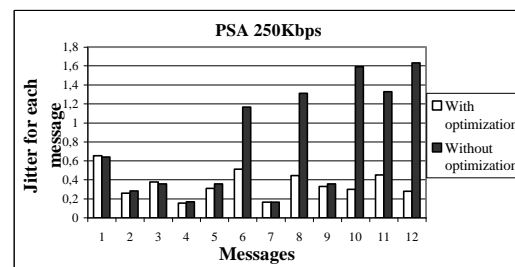


Figure 5 - Results for PSA benchmark at 250 kbps.

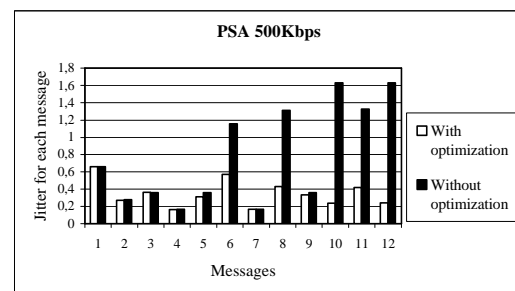


Figure 6 - Results for PSA benchmark at 500 kbps.

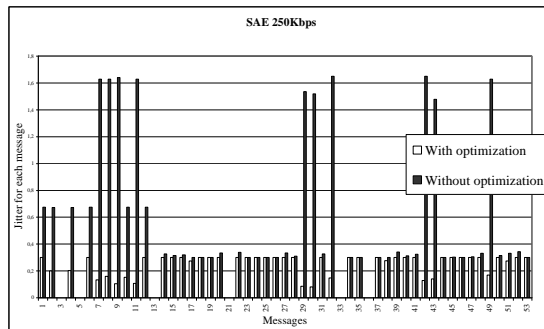


Figure 7 – Results for SAE benchmark at 250 kbps.

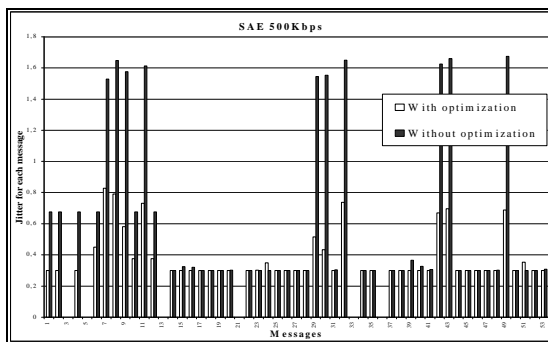


Figure 8 - Results for SAE benchmark at 500 kbps.

These results show a clear improvement on the message jitter for at least some of the messages in the set. These improvements are quite large for some messages and very small or non-existing for others.

On most cases, increased jitter was discovered for greater bandwidths. It is not clear why this is the case, but it would appear that the scheduling optimization effort needs to be higher for those settings. We'll try to address this issue by designing and evaluating new transformations, and possibly using different optimization algorithms.

Even so, jitter for optimised solutions was always reduced by over 50% in the PSA message set and around 40% in the SAE set.

## 5. CONCLUSIONS

We developed a scheduling and optimization tool capable of building the so-called system matrix for a TTCAN based network given a set of periodic messages. The scheduling always generates a valid solution for feasible sets, and the optimization phase refines this solution so that message jitter is reduced. Results are satisfactory: the tool generates valid scheduling tables for feasible benchmarks and the optimization phase in all cases enhanced those solutions by providing additional jitter

reduction. For the two message sets tested, the PSA and SAE benchmarks, the optimization was more effective at lower transmission rates. This issue needs further investigation in order to introduce additional improvements in the optimization process.

## REFERENCES

- Kopetz, H. (1997). Real Time Systems Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers.
- Tindel, K., Burns, A. (1994). Guaranteeing Message Latencies on Control Area Network (CAN). *Proceedings of the ICC'94*. Mainz, Germany.
- Navet, N., Song, Y.-Q. (1997). Performance and Fault Tolerance of Real Time Applications Distributed over CAN. CiA – CAN in Automation Research Award.
- Führer, T., Bernd Müller, Werner Dieterle, Florian Hartwich, Robert Hugel, Michael Walther (2000). Time-triggered Communication on CAN (Time-triggered CAN-TTCAN). *Proceedings of the 7<sup>th</sup> international CAN conference*.
- Hartwich, F., Bernd Müller, Thomas Führer, Robert Hugel (2000). CAN Network with Time-triggered Communication. *Proceedings of the 7<sup>th</sup> international CAN conference*.
- Coutinho, F., J. Fonseca, J. Barreiros, E. Costa (2000). Using Genetic Algorithms to Reduce Jitter in Control Variables Transmitted over CAN. *Proceedings of the 7<sup>th</sup> international CAN conference*.
- Michalewicz, Z. (1999). Genetic Algorithms + Data Structures = Evolution Programs (3<sup>rd</sup>, revised and extended edition), Springer-Verlag. Berlin.